การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง

ADAPTIVE MULTI-SHELL MEDIAN FILTER

นฤพนธ์ พนากุลชัยวิทย์
NARUPON PANAKULCHIWIT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2543

ISBN 974-622-649-5

การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง

ADAPTIVE MUTI-SHELL MEDIAN FILTER

นฤพนธ์ พนากุลชัยวิทย์

NARUPON PANAKULCHIWIT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2543

ISBN 974-622-649-5

ADAPTIVE MUTI-SHELL MEDIAN FILTER

NARUPON PANAKULCHIWIT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LARDKRABANG

2000

ISBN 974-622-649-5

หัวข้อวิทยานิพนธ์	การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง
นักศึกษา	นฤพนธ์ พนากุลชัยวิทย์
รหัสประจำตัว	36061037
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2543
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.กอบชัย เดชหาญ

บทคัดย่อ

ปัญหาหลักของงานประมวลผลทางภาพ คือการเกิดสัญญาณรบกวนแบบขาวดำ ซึ่งจะทำให้คุณค่าหรือความคมชัดของภาพสูญเสียไป การกรองมัธยฐานได้ถูกนำมาเสนอขึ้นมา เพื่อช่วยปรับปรุงคุณภาพของภาพให้ดีขึ้น แต่ก็ยังทำให้ภาพมีความเบลอสูง เพราะว่ารายละเอียดเล็กๆ น้อยๆ ได้สูญเสียไปกับการกรองมัธยฐาน วิทยานิพนธ์ฉบับนี้จึงได้นำเสนอวิธีการกรองมัธยฐานแบบดัดแปลง เพื่อใช้ปรับปรุงคุณภาพให้มีความคมชัดสูง โดยสามารถกำจัดสัญญาณรบกวนแบบขาวดำได้มาก โดยมีวิธีการที่ง่าย และให้ผลที่ดีกว่าการกรองสัญญาณมัธยฐานแบบทั่วๆ ไป เช่น แบบขนาดหน้าต่างคงที่

Thesis Title	Adaptive Multi-Shell Median Filter
Student	Mr. Narupon Panakulchiwit
Student ID.	36061037
Degree	Master of Engineering
Programme	Electrical Engineering
Year	2000
Thesis Advisor	Assoc.Prof.Dr. Kobchai Dejhan

ABSTRACT

The picture of image is disturbed by impulse noise on image pre-processing which causes the loss quality and fidelity. The median filter is proposed to improve the impulse noise but the image has more blurring because of the loss of fine detail on median filter processing. This thesis proposes an adaptive multi-shell median filter to improve impulse noise to obtain the better results for image processing. The performances of the proposed filter can be compared with the regular median filters or fixed length.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ ก็เพราะได้รับความกรุณาจาก รศ. ดร. กอบชัย เดชหาญ ผู้ซึ่งคอยดูแลให้คำแนะนำในการทำวิจัยแก่ผู้วิจัยมาโดยตลอด จึงขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้ นอกจากนี้ยังรวมถึง รศ. ดร. ฟูศักดิ์ ชีวสุวิทย์ อ. สักรียา ชิตวงศ์ อ.อาโมทย์ สมบูรณ์แก้ว ที่ได้ให้ความกรุณาและคอยติดตามความคืบหน้าตลอดเวลา

ขอขอบพระคุณบริษัท NEC แห่งประเทศไทยที่ได้ให้การสนับสนุนทุนการศึกษาคอมพิวเตอร์และการสื่อสาร ประจำปีการศึกษา 2536 ขอขอบคุณ คุณอดิศักดิ์ แข็งสาริกิจ และคุณจิรวุฒิ สินธุวณิชเศรษฐ์ ที่คอยให้คำปรึกษารวมถึงเป็นเพื่อนที่ดีมาตลอดเวลาในการทำวิทยานิพนธ์

สุดท้ายนี้ผู้ทำวิจัยขอขอบคุณ คุณรุ่งตะวัน พนากุลชัยวิทย์และน้องผิงผิง รวมถึงคุณแม่และพี่ ๆ ที่เป็นกำลังใจที่ดีเสมอมา

นฤพนธ์ พนากุลชัยวิทย์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

เทคโนโลยีทางด้านไมโครคอมพิวเตอร์ได้ก้าวหน้าไปอย่างรวดเร็ว จากคอมพิวเตอร์แบบ 4 บิทในยุคแรกมาเป็นไมโครคอมพิวเตอร์ เช่น คอมพิวเตอร์ขนาด 16 บิทของบริษัทไอบีเอ็มที่ใช้ชิปไอซีตระกูล 80X86 เป็นที่ได้รับความนิยมอย่างสูง ปัจจุบันไมโครคอมพิวเตอร์จะต้องใช้ชิปเพนเทียม 200 เมกะเฮิรตซ์ เอ็มเอ็มเอ็กซ์ (Pentium 200 MHz/MMX) เป็นอย่างน้อย ในอนาคตคาดการณ์ว่าทุก ๆ บ้าน จะต้องมีคอมพิวเตอร์ใช้ อย่างน้อย 1 เครื่อง นอกจากนี้เครื่องคอมพิวเตอร์ส่วนมากในโลกนี้ก็จะต่อถึงกันโดยผ่านโมเด็ม (Modem) ไปยังเครือข่ายอินเตอร์เน็ต

ศาสตร์ทางด้านการประมวลผลสัญญาณภาพ (Digital image processing) ได้เข้ามามีบทบาทอย่างมากต่อการประยุกต์ใช้งานของไมโครคอมพิวเตอร์ในด้านต่าง ๆ ทั้งการพาณิชย์ การอุตสาหกรรม ความมั่นคง ด้านการศึกษา การสำรวจทรัพยากรธรรมชาติ ด้านการแพทย์ เป็นต้น ข้อมูลข่าวสารต่าง ๆ จะอยู่ในรูปของภาพ เสียง เอกสาร สัญลักษณ์ ซึ่งคอมพิวเตอร์จะไม่สามารถรับรู้สิ่งเหล่านี้ได้โดยตรง ในขบวนการเริ่มแรกของการประยุกต์ใช้ศาสตร์ทางด้านนี้ จะต้องมีการแปลงและเก็บข้อมูลเข้ามาอยู่ในรูปของสัญญาณเชิงเลข (Digital) จากอุปกรณ์อินพุทของการใช้งานในด้านนั้นๆ เช่น ไมโครโฟน กล้องถ่ายภาพ กล้องวิดีโอ สัญญาณโทรทัศน์ เครื่องอ่านฟิล์ม ดาวเทียม เครื่อง MRI เครื่อง CT-SCAN ULTRA-SOUND เป็นต้น โดยในวิทยานิพนธ์เป็นการศึกษาถึงการแก้ปัญหาสัญญาณรบกวนที่เกิดขึ้นในสัญญาณภาพ เนื่องจากสัญญาณภาพส่วนมากล้วนเป็นสัญญาณที่มีรายละเอียดสูง แต่มีขนาด (Amplitude) ของสัญญาณต่ำ จึงมีโอกาสในการถูกรบกวนได้สูงจากสภาวะแวดล้อม และในขบวนการของการแปลงจากสัญญาณภาพ (Analog) เป็นสัญญาณเชิงเลข จะต้องใช้เครื่องมือหรือฮาร์ดแวร์ในการทำงาน ในขั้นตอนนี้จะมีโอกาสในการเกิดสัญญาณรบกวนได้ง่ายและส่วนใหญ่เป็นสัญญาณรบกวนแบบขาวดำ (Impulse noise) ดังนั้นก่อนที่จะนำข้อมูลเหล่านี้ไปประมวลผลในขบวนการต่อไป จะต้องมีการเตรียมหรือปรับแต่งข้อมูลเหล่านั้นให้มีความสมบูรณ์ก่อน วิธีการส่วนใหญ่ที่นิยมใช้ คือ นำข้อมูลภาพมาผ่านการกรองมัธยฐาน (Median filter) เพื่อกำจัดสัญญาณรบกวนที่ไม่ต้องการให้หมดไป

ที่ผ่านมาในอดีต ได้มีผู้ทำวิจัยเกี่ยวกับเรื่องการกรองมัธยฐานเป็นจำนวนมาก โดยการนำเสนอวิธีการกรองมัธยฐานหลาย ๆ วิธีการด้วยกัน ซึ่งแต่ละแบบก็จะมีข้อดี ข้อเสียแตกต่างกันออกไป ในวิทยานิพนธ์นี้ได้นำเสนอการกรองมัธยฐานแบบมีประสิทธิภาพสูง ความซับซ้อนต่ำและสามารถนำไปประยุกต์หรือดัดแปลงเพื่อใช้งานต่อไปได้อย่างสะดวก

1.1 วัตถุประสงค์ของวิทยานิพนธ์

โดยปกติบุคคลโดยทั่ว ๆ ไปนั้นจะไม่ต้องการภาพที่ไม่คมชัดหรือภาพที่มีสัญญาณรบกวน ภาพที่มีสัญญาณรบกวนจะไม่เป็นที่ต้องการแก่บุคคลที่ได้พบเห็น ถ้าเราสามารถหาวิธีจัดการสัญญาณรวบกวนออกไปได้ก็ย่อมทำให้ภาพนั้นมีคุณค่าขึ้นมาไม่มากก็น้อย ยิ่งถ้าวิธีการนั้นสามารถกำจัดสัญญาณรบกวนได้ดีและยังสามารถรักษาความคมชัดของภาพได้ก็จะเป็นที่ต้องการอย่างมาก

สัญญาณรบกวนแบบอิมพัลส์เกิดขึ้นได้หลายลักษณะและตลอดเวลา ทั้งจากธรรมชาติและฝีมือมนุษย์ เนื่องจากสัญญาณภาพโดยทั่วไปมีขนาดแอมปลิจูดต่ำมาก (ประมาณ 0.8 โวลท์ในระบบโทรทัศน์) จึงถูกรบกวนได้โดยง่าย วิธีการกรองสัญญาณแบบมัธยฐานจะมีความเหมาะสมในการกำจัดสัญญาณรบกวนลักษณะนี้และมีคุณสมบัติในการรักษาขอบภาพได้ดี ดังจะได้กล่าวถึงรายละเอียดอีกครั้งในบทที่ 3 การกรองมัธยฐานได้มีผู้นำเสนอและคิดค้นไว้หลายแบบ ในส่วนของวิทยานิพนธ์นี้จะนำเสนอการกรองมัธยฐานมัลติเชลล์แบบดัดแปลง ซึ่งจะสามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์ได้ดี ให้ความคมชัดของภาพสูงและมีวิธีการคำนวณที่ไม่ยุ่งยากและซับซ้อน งานวิจัยนี้คาดว่าจะเป็นประโยชน์ต่อการวิจัยและพัฒนาในสาขาการประมวลภาพเบื้องต้น งานประมวลภาพผลทางด้านอื่นที่เกี่ยวข้อง นอกจากนี้ยังรวมไปถึงการประยุกต์ใช้ในภาพถ่ายทางไกลจากดาวเทียมด้วย

1.2 ขอบเขตของการวิจัย

งานวิจัยนี้จะได้นำเสนอถึงหลักการและวิธีการกรองมัธยฐานแบบดัดแปลง เพื่อแสดงให้เห็นถึงข้อดีและข้อเสียรวมถึงประโยชน์ที่จะได้รับ โดยเนื้อหาได้กล่าวถึงวิธีการที่ได้เคยมีผู้ทำวิจัยไว้แล้วและส่วนที่ผู้ทำวิจัยได้ทำขึ้นใหม่ เพื่อที่จะได้ชี้ให้เห็นถึงข้อดีและข้อเสียของแต่ละวิธี ทฤษฎีที่จะใช้เพื่อเปรียบเทียบภาพผลลัพธ์ ได้แก่ ค่าเฉลี่ยค่าผิดพลาดกำลังสอง (MSE) และอัตราส่วนสัญญาณต่อสัญญาณรบกวน (SNR) โดยรายละเอียดของเนื้อหาในวิทยานิพนธ์นี้สามารถแบ่งได้ดังนี้

บทที่ 1 เป็นบทนำ กล่าวถึงวัตถุประสงค์ และขอบเขตของวิทยานิพนธ์

บทที่ 2 เนื้อหาเกี่ยวกับระบบพื้นฐานของการประมวลภาพ ทฤษฎีและวิธีการกำเนิดสัญญาณรบกวนแบบอิมพัลส์ รวมถึงทฤษฎีที่ใช้ในการวัดเพื่อเปรียบเทียบคุณภาพของภาพ

บทที่ 3 อธิบายถึงหลักการกรองมัธยฐานเบื้องต้น ข้อดีและข้อเสียของวิธีนี้ นอกจากนี้ได้มีการกล่าวถึงวิธีการกรองมัธยฐานที่เคยมีการวิจัยไว้แล้ว รวมถึงวิธีการกรองมัธยฐานแบบมัลติเชลล์

บทที่ 4 อธิบายหลักการทำงานของวิธีการกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ

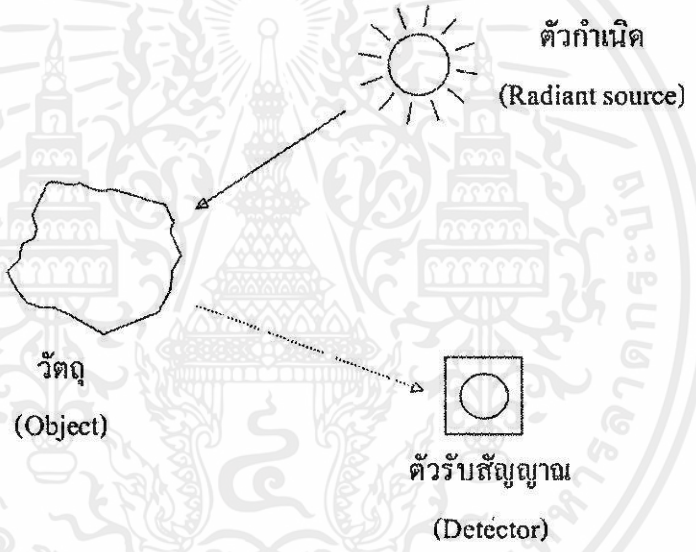
บทที่ 5 อธิบายหลักการทำงานการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ผู้วิจัยได้คิดค้นขึ้นทั้ง 2 แบบ นอกจากนี้ยังได้เปรียบเทียบกับการกรองมัธยฐานชนิดอื่น ๆ ด้วย

บทที่ 6 จะเป็นเนื้อหาเกี่ยวกับการวิเคราะห์วิจารณ์ผลการทดลองทั้งหมด สรุปผล พร้อมทั้งนำเสนอแนวทาง ปัญหา ข้อคิดเห็น เพื่อเป็นการพัฒนาต่อไปในอนาคต

ส่วนภาคผนวกจะเป็นเนื้อหาโปรแกรมทั้งหมดที่เกี่ยวข้องกับงานวิจัยนี้ที่เขียนขึ้นโดยการใช้โปรแกรมภาษาซีของบริษัทบอร์แลนด์ เวอร์ชัน 3.1

บทที่ 2
ทฤษฎีพื้นฐานในการประมวลผลสัญญาณภาพ

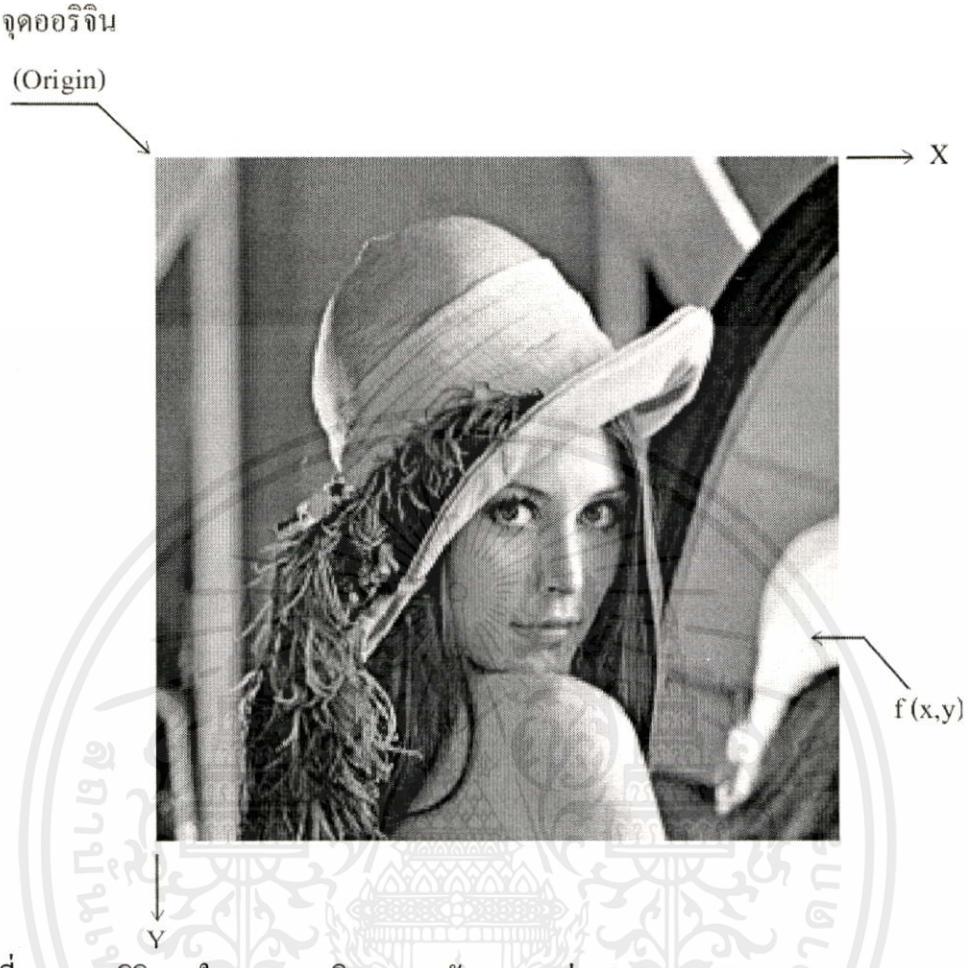
การที่มนุษย์สามารถมองเห็นสิ่งต่าง ๆ ได้นั้น เกิดจากแสงที่ตกกระทบต่อพื้นผิววัตถุแล้วสะท้อนเข้าตา จำนวนของความเข้มแสงที่แตกต่างกันได้ก่อให้เกิดเป็นภาพขึ้น เช่นเดียวกับหลักการของกล้องถ่ายภาพ คือ ภาพเกิดจากความเข้มแสงที่ตกกระทบฟิล์มมีความแตกต่างกัน เมื่อนำฟิล์มไปล้างก็จะได้ภาพที่ปรากฏบนฟิล์ม วิธีการสร้างภาพนอกจากที่เกิดจากแสงตามธรรมชาติแล้วยังมีอีกหลายวิธี เช่น เลเซอร์ หลอดรังสีเอ็กซ์ ความร้อน คลื่นเสียง ทำหน้าที่เป็นตัวกำเนิด (radiant source) ซึ่งแต่ละวิธีก็จะต้องมีตัวรับสัญญาณ (detector) ที่เหมาะสมแตกต่างกัน



ภาพที่2.1 วิธีการสร้างภาพที่ประกอบด้วยองค์ประกอบหลักคือตัวกำเนิด วัตถุ และตัวรับสัญญาณ

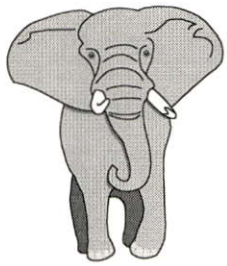
2.1 ระบบพื้นฐานการประมวลสัญญาณภาพดิจิตอล

สัญญาณภาพที่สายตามนุษย์สามารถมองเห็น และสัญญาณภาพที่ใช้ในการส่งออกอากาศทางโทรทัศน์ รวมถึงสัญญาณภาพจากกล้องวิดีโอเหล่านี้ล้วนจัดเป็นสัญญาณอนาลอก (analog signal) ซึ่งไม่เหมาะสมสำหรับการนำไปประยุกต์ใช้กับงานด้านการประมวลผลสัญญาณภาพดิจิตอล (digital image processing) ดังนั้นสัญญาณดังกล่าวต้องถูกแปลงเป็นสัญญาณดิจิตอลเสียก่อน โดยเครื่องแปลงสัญญาณภาพเป็นสัญญาณดิจิตอล (Image digitizer or frame grabber) จะได้ภาพที่มีขนาด 2 มิติ f(x,y) เป็นฟังก์ชันของระดับความเข้มแสงบนภาพในตำแหน่ง x และ y ในระบบสปาเชียลโคออร์ดิเนท ค่าของ f บน (x,y) ใด ๆ จะเป็นสัดส่วนของระดับความสว่าง (gray level) ของภาพที่จุดนั้น

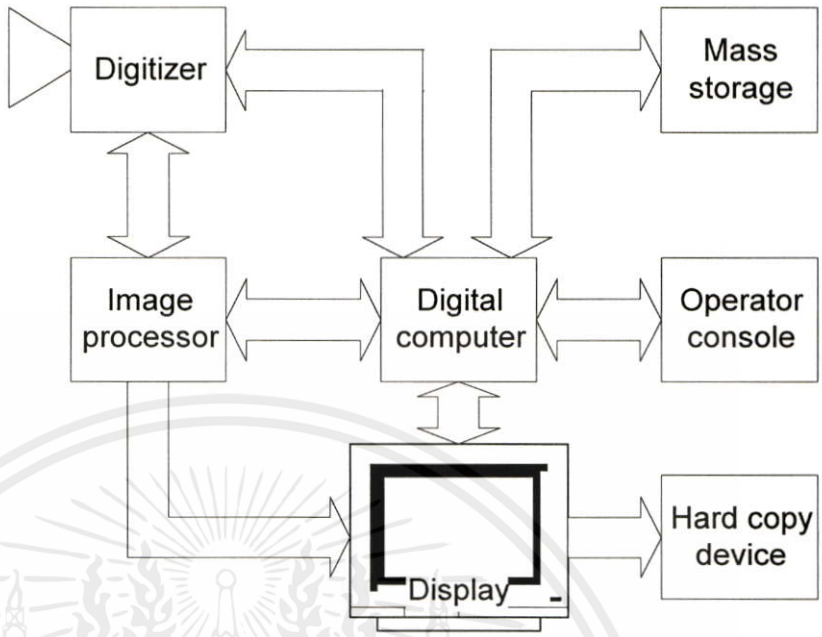


ภาพที่ 2.2 ภาพดิจิตอลในระบบ 8 บิทของระดับความสว่าง (grey level)

การกำหนดแกน x,y ในการอ้างอิงจะใช้วิธีที่สอดคล้องกับโปรแกรมภาษาซี เช่น เทอร์โบซี 2.0 หรือบอร์แลนด์ซีพลัสพลัส 3.1 ของบริษัทบอร์แลนด์ โดยมีจุดออริจินที่มุมบนซ้าย ค่าแกน x วิ่งไปทางขวา ค่าแกน y วิ่งลงล่างตรงนี้จะมีความแตกต่างกับลักษณะกราฟทั่ว ๆ ไปที่ค่า y จะวิ่งขึ้นข้างบน เหตุผลที่มีการกำหนดดังนี้เพื่อให้สอดคล้องกับการแสกนของลำอิเล็กตรอนในระบบจอภาพของโทรทัศน์ที่มีการแสกนจากซ้ายไปขวาและจากบนลงล่าง ระบบการประมวลผลสามารถอธิบายได้ดังรูปนี้



Image



ภาพที่ 2.3 ระบบการประมวลผลสัญญาณภาพดิจิตอล

ชุด digitizer จะประกอบไปด้วยกล้อง หรืออุปกรณ์รับสัญญาณภาพ และโมดูลของการแปลง อนาลอกเป็นดิจิตอล สัญญาณภาพดิจิตอลที่ได้จะถูกส่งไปประมวลผลที่ image processor หรืออาจจะเป็นตัว digital computer เพื่อทำการประมวลผลและเก็บข้อมูลภาพทั้งหมดนั้น ไว้ในรูปของหน่วยความจำสำรอง (mass storage device)

การแปลงอนาลอกเป็นดิจิตอลจะเป็นขนาดความละเอียดของระดับเทา ซึ่งจะเท่าใดก็ได้ขึ้นอยู่กับความต้องการ และขีดความสามารถของเทคโนโลยี รวมถึงราคาที่เหมาะสมในยุคนั้น ๆ เนื่องจากสัญญาณภาพมีรายละเอียดของข้อมูลมาก การแซมปลิ้งเพื่อที่จะเก็บข้อมูล ต้องมีความรวดเร็วพอ ไอซีที่จะใช้ทำหน้าที่นี้ต้องเป็น flash A/D เช่น CA3318 ของ Harris เมื่อประมาณปี พ.ศ. 2534 ไอซีประเภท ขนาด 8 บิท มีราคาประมาณ 5,000 บาท ในขณะที่ปัจจุบันราคาได้ลดลงไปมาก เหลือแค่ประมาณ 600 บาท ส่วน flash A/D ขนาด 6 บิท ก็จะมีราคาถูกกว่าขนาด 8 บิทเป็นอย่างมาก โดยจำนวนระดับของค่าระดับเทา สามารถแสดงได้ดังสมการ 2.1

$$G=2^m \qquad (2.1)$$

เมื่อ G คือจำนวนระดับของ gray level

m คือ จำนวนบิทในการแปลง A/D

สำหรับภาพที่ผ่านการแปลงแล้วจะถูกเก็บอยู่ในหน่วยความจำในรูปแบบของแอเรย์ 2 มิติ โดยมีขนาดของภาพเป็น MXN จำนวนจุดภาพ ดังสมการ 2.2

$$f(x,y)=\begin{bmatrix} f(0,0) & f(1,0) & \cdots & f(M-1,0) \\ f(0,1) & f(1,1) & \cdots & f(M-1,1) \\ \vdots & & & \\ f(0,N-1) & f(1,N-1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (2.2)$$

เมตริกซ์ทางขวามือสมการคือภาพดิจิตอล โดยแต่ละองค์ประกอบในเมตริกซ์จะเป็น image element หรือ picture element อาจเรียกโดยชื่อย่อว่า pel หรือ pixel ก็ได้ ขนาดของภาพ คือ MxN ที่นิยมโดยทั่วไป เช่น 256x256 หรือ 512x512

2.2 ทฤษฎีการเปรียบเทียบคุณภาพของภาพ

ในการประมวลผลภาพนั้น ถึงแม้ว่าผลลัพธ์ที่ได้อาจจะดีขึ้นหรือแย่ลงก็ตาม แต่เราสามารถเปรียบเทียบคุณภาพของภาพได้ด้วยสายตาระหว่างภาพต้นฉบับกับภาพที่ผ่านการประมวลผลแล้ว แต่วิธีการนี้จะไม่สามารถบอกคุณภาพของภาพที่ได้ออกมาเป็นตัวเลขได้ว่าภาพนี้ดีขึ้นหรือแย่ลงเป็นกี่เปอร์เซ็นต์ของภาพที่ต้องการเปรียบเทียบ วิธีที่ใช้เปรียบเทียบเพื่อให้ออกมาในรูปของตัวเลขหรือเพื่อให้อยู่ในรูปแบบมาตรฐานการเปรียบเทียบ เช่น การค่ารากที่สองของค่าเฉลี่ยผิดพลาดกำลังสอง
(root mean square error ; RMSE) ค่าเฉลี่ยผิดพลาดกำลังสอง (mean square error ; MSE) ค่าเฉลี่ยผิดพลาดทางขนาด (mean absolute error ; MAE) ในวิทยานิพนธ์นี้จะใช้ค่า MSE เป็นหลัก

วิธี MSE ได้ถูกคิดค้นขึ้นโดย Justusson [2] ซึ่งได้รับความนิยมอย่างสูงจากผู้ที่ทำงานวิจัยด้านการประมวลผลภาพและวิธีการนี้ยิ่งได้ถูกอ้างอิงถึงจากบทความอื่น ๆ อีกหลายบทความ

กำหนดให้

$S_{i,j}$ คือค่าระดับสีเทาของภาพต้นแบบที่จุด i,j

$X_{i,j}$ คือค่าระดับสีเท่าของภาพที่ผ่านการประมวลผลที่จุด i,j

M,N คือขนาดของภาพทางแนวนอนและแนวตั้งตามลำดับ

ค่าเฉลี่ยผิดพลาดกำลังสอง (Mean square error:MSE) มีค่าตามสมการที่ 2.3

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(S_{i,j} - X_{i,j}\right)^2 \qquad (2.3)$$

ค่าเฉลี่ยผิดพลาดกำลังสอง MSE จะใช้วัดความแตกต่างของข้อมูล ภาพค่า MSE ถ้ามีค่าน้อยจะเป็นดัชนีบอกว่าภาพนั้นมีคุณภาพดี ในทำนองตรงข้ามกันถ้าค่า MSE มากแสดงว่าภาพนั้นมีคุณภาพที่ไม่ดี

ค่าเฉลี่ยผิดพลาดทางขนาด (Mean absolute error : MAE) มีค่าตามสมการที่ 2.4

$$MAE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left|S_{i,J} - X_{i,J}\right| \qquad (2.4)$$

ค่าเฉลี่ยผิดพลาดทางขนาดเป็นการหาค่าเฉลี่ยทางขนาดของค่าความแตกต่างระหว่างภาพต้นแบบกับภาพที่ถูกประมวลผลโดยตรง จะใช้วัดกับข้อมูลภาพที่มีความแตกต่างกันมาก ๆ MAE มีคุณสมบัติเป็นฟังก์ชันเชิงเส้น ดังนั้นค่านี้มีค่ายิ่งน้อยก็ยิ่งดีในทำนองเดียวกับค่า MSE

สำหรับค่ารากที่สองของค่าเฉลี่ยผิดพลาดกำลังสอง (Root mean square error : RMSE) มีค่าตามสมการที่ 2.5

$$RMSE = \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(S_{i,j} - X_{i,j}\right)^2} \qquad (2.5)$$

การหาค่ารากที่สองของค่าเฉลี่ยผิดพลาดกำลังสอง นิยมใช้วัดคุณภาพของภาพในงานบีบอัดและขยายข้อมูลภาพ ค่านี้จะคล้ายกับ MSE ส่วนวิธีการใช้แต่ละแบบนั้นขึ้นอยู่กับจุดประสงค์ของงานและความเหมาะสมของงานนั้น ๆ

อัตราส่วนสัญญาณต่อสัญญาณรบกวน (Signal to noise ratio : SNR) มีค่าตามสมการที่ 2.6

$$SNR = 10\log_{10}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}X_{i,j}^2}{\sum_{i=1}^{M}\sum_{j=1}^{N}\left(S_{i,j} - X_{i,j}\right)^2} \qquad (2.6)$$

การหาค่าอัตราส่วนของสัญญาณต่อสัญญาณรบกวน มีหน่วยเป็นเดซิเบล (dB) ค่า SNR ยิ่งมากจะยิ่งดีมากเพราะนั้นหมายว่าค่าพลังงานของข้อมูลภาพมีมากกว่าพลังงานของสัญญาณรบกวน

2.3 สัญญาณรบกวนแบบอิมพัลส์ (Impulse noise)

เป็นสัญญาณรบกวนที่มีขนาดของแอมปลิจูดสูงมากและมีความกว้างของสัญญาณแคบมาก ขนาดของสัญญาณมีโอกาสเป็นไปได้ทั้งบวกและลบ โดยมีความน่าจะเป็นเท่า ๆ กัน สัญญาณรบกวนชนิดนี้เกิดขึ้นได้จากการกระชากอย่างรุนแรงของระบบไฟฟ้ากำลังสูง ๆ การผิดพลาดของบิทข้อมูลดิจิตอลในกระบวนการสื่อสาร หรือเกิดจากการผิดพลาดของการแปลงจาก ข้อมูลอนาลอกเป็นดิจิตอล (Analog to digital malfunction)

สมการของการเกิดอิมพัลส์นอยส์มีค่าตามสมการที่ 2.7

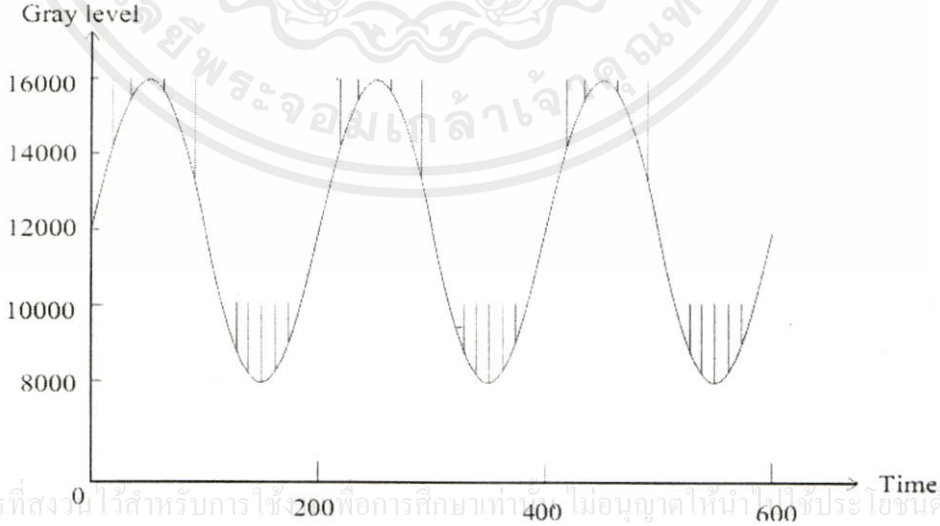
$$N_{i,j}^{imp} = \begin{cases} d & \text{เมื่อ } P^{+}/2 \\ -d & \text{เมื่อ } P^{-}/2 \\ 0 & \text{เมื่อ 1-P} \end{cases} \qquad (2.7)$$

$N_{i,j}^{imp}$ คือค่าระดับสีเท่าของภาพที่เกิดสัญญาณรบกวนแบบอิมพัลส์ที่จุด i, j

d คือขนาดของสัญญาณรบกวนแบบอิมพัลส์

P คือความน่าจะเป็นของการเกิดสัญญาณรบกวน

+,- แสดงสัญญาณรบกวนช่วงค่าบวก (salt noise) และช่วงค่าลบ (pepper noise)



ภาพที่ 2.4 สัญญาณรบกวนแบบอิมพัลซ์ที่ปรากฏบนสัญญาณซายน์เวฟ

สัญญาณรบกวนแบบอิมพัลส์เกิดขึ้นกระจายอยู่ทั่วไปในภาพขึ้นอยู่กับความน่าจะเป็นของการเกิดสัญญาณรบกวน P ซึ่งมีฟังก์ชันการกระจายแบบยูนิฟอร์ม

ภาพที่ 2.5 ภาพ LENA ที่มีสัญญาณรบกวนอิมพัลส์ 10 %

2.5 สัญญาณรบกวนแบบขาดหายไปของเส้นภาพ (Missing line noise)

สัญญาณรบกวนแบบขาดหายไปของเส้นภาพ (Missing line noise) ในกระบวนการส่งสัญญาณเพื่อการสื่อสาร หรือถ่ายทอดสัญญาณโทรทัศน์ อาจเกิดการสูญเสียหรือขาดหายไปของเส้นสัญญาณภาพบางเส้น ซึ่งไม่สามารถแก้ไขให้ดีกลับคืนมาได้ การสูญเสียหรือขาดหายไปของเส้นสัญญาณภาพบางเส้นดังกล่าวเรียกว่าเป็นสัญญาณรบกวนแบบขาดหายไปของเส้นภาพ (missing line noise) [13] และการขาดหายของเส้นสัญญาณภาพจะเกิดเฉพาะในแนวนอนเท่านั้น เนื่องจากระบบการส่งสัญญาณของโทรทัศน์จะใช้วิธีในการสแกนแนวนอนก่อนหลังจากนั้นก็จะทำการสแกนทีละเส้นแล้วจึงไล่จากบนลงล่าง ดังนั้นจึงอาจมีการขาดหายไปของสัญญาณได้ถ้าถูกรบกวนหรือมีข้อผิดพลาดในขณะการทำงาน

ภาพที่ 2.6 ภาพ LENA ที่มีสัญญาณรบกวนแบบ Missing line noise และ อิมพัลส์ 5 %

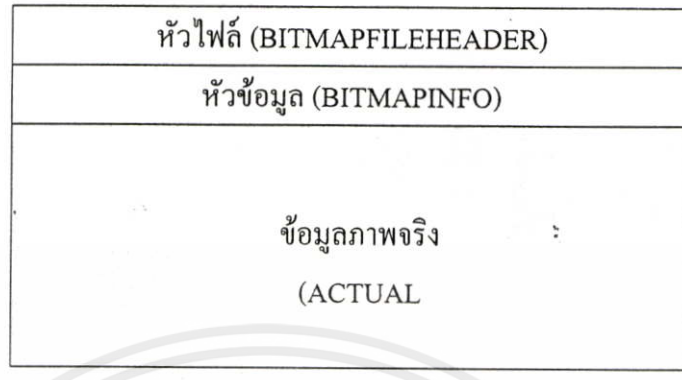
2.6 รูปแบบของข้อมูลภาพถ่ายโดยทั่วไป

รูปแบบของข้อมูลที่มีใช้อยู่โดยทั่วไปนั้น เช่น GIF, IFF/LBM, WPG, PIC, TGA, PCX และ TIFF เป็นต้น รูปแบบของข้อมูลที่นิยมใช้กันมาก คือ PCX, GIF, BMP และ TIFF สำหรับรูปแบบของข้อมูลแบบ BMP หรือบิตแมพจัดเป็นรูปแบบที่ได้รับความนิยมมากที่สุด เพราะข้อมูลแบบบิตแมพนี้ไม่มีการบีบข้อมูล (compression) จึงมีความรวดเร็วและสะดวกในการอ่านข้อมูลอย่างมาก นอกจากนี้รูปแบบของข้อมูลกลางที่ใช้ในการแลกเปลี่ยนระหว่างโปรแกรมประยุกต์ต่าง ๆ ใช้ในการส่งข้อมูลไปให้อุปกรณ์แสดงผล และอุปกรณ์อื่น ๆ เป็นต้น แต่ในรูปแบบบิตแมพนี้มีข้อจำกัดคือต้องเปลืองเนื้อที่ฮาร์ดดิสในการจัดเก็บข้อมูลสูงกว่ารูปแบบอื่นที่ใช้การเข้ารหัสบีบข้อมูล

2.6.1 รูปแบบของข้อมูลภาพ BMP

รูปแบบของข้อมูล BMP ใช้กับโปรแกรมระบายสีในวินโดว์เวอร์ชัน 3.0 เป็นต้นมา โดยรูปแบบของข้อมูลแบบนี้สามารถเป็นข้อมูลสีได้ ตั้งแต่ 1 ถึง 24 บิท มีส่วนหัว (header) ในการบอกรายละเอียดต่าง ๆ ของภาพ โดยกำหนดในลักษณะโครงสร้าง (structure) ในภาษาระดับสูง เช่น ภาษาซี ข้อมูลภาพในรูปแบบมิตแมพแบ่งออกได้ 3 ส่วน คือ หัวไฟล์ (BITMAPFILEHEADER) ตามตารางที่ 2.1 หัวข้อมูล (BITMAPINFO) ตามตารางที่ 2.2 และ 2.3 ซึ่งรวมทั้งข้อมูลต่าง ๆ และพาเลตของสี (color palette) และ ข้อมูลภาพจริง (actual image) จากภาพที่2.7 ข้อมูลภาพในรูปแบบ

BMP นั้นแสดงส่วนประกอบของโครงสร้างที่ใหญ่ที่สุดของภาพ ซึ่งข้อมูลพาเลตสีและข้อมูลภาพจะเปลี่ยนโครงสร้างไปตามรูปแบบของจำนวนสีของภาพและวิธีการถอดรหัส



ภาพที่ 2.7 โครงสร้างที่ใหญ่ที่สุดของภาพในรูปแบบบิทแมพ

ตารางที่ 2.1 รายละเอียดเกี่ยวกับหัวไฟล์ของภาพบิทแมพ (BITMAPFILEHEADER)

ขนาด	ชื่อตัวแปร	ความหมาย
-WORD	BfType;	เป็นชนิดของรูปแบบข้อมูลภาพสำหรับบิทแมพ คือ BM
-DWORD	BfSize;	เป็นขนาดของไฟล์ทั้งหมด
-WORD	BfReserved 1;	ไม่ได้ใช้งาน ปกติกำหนดให้มีค่าเป็นศูนย์
-WORD	BfReserved 2;	ไม่ได้ใช้งาน ปกติกำหนดให้มีค่าเป็นศูนย์
-DWORD	bfOffBits;	เป็นขนาดของหัวไฟล์ทั้งหมด ใช้เพื่อข้ามไปจุดเริ่มต้นของข้อมูลภาพจริง

ตารางที่ 2.2 องค์ประกอบของหัวข้อมูล (BITMAPINFO)

หัวข้อมูล	ลักษณะ	ความหมาย
-BITMAPINFOHEADER	BmiHeader;	เป็นข้อมูลเกี่ยวกับขนาดต่าง ๆ จำนวนสี และรูปแบบของสี
-RGBQUAD	bmiColors (1);	เป็นอาร์เรย์ข้อมูลสีอาร์จีบีที่เป็นตัวกำหนดสีในการแสดงผลของภาพ

ตารางที่ 2.3 องค์ประกอบโครงสร้างของ BITMAPINFOHEADER

ขนาดของพารามิเตอร์	ชื่อพารามิเตอร์	ความหมาย
-DWORD	BiSize;	เป็นขนาดของส่วนหัวข้อมูลมีหน่วยเป็นไบต์
-DWORD	BiWidth;	เป็นความกว้างของภาพ มีหน่วยเป็น จุดภาพ/เส้น
-DWORD	BiHeight	เป็นความสูงของภาพ มีหน่วยเป็นเส้น
-WORD	BiPlanes;	เป็นจำนวนหน้าของสี (color plane) สำหรับอุปกรณ์เป้าหมาย (targetdevice) ปกติกำหนดให้เป็น 1 เสมอ
-WORD	BiBitCount;	จำนวนบิทต่อจุดภาพ (1, 4, 8, 24) *
-DWORD	biCompression;	เป็นชนิดของการบีบอัด**
-DWORD	biSizeImage;	เป็นขนาดของภาพมีหน่วยเป็นไบต์
-DWORD	biXPelsPerMeter;	เป็นความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวนอนต่อหนึ่งเมตร
-DWORD	biYPelsPerMeter;	เป็นความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวตั้งต่อหนึ่งเมตร
-DWORD	BiClrUsed;	เป็นจำนวนดัชนีสี (color index) ในตารางสี (color table) มีค่าได้ 3 กรณี***
-DWORD	BiClrImportant;	เป็นจำนวนความสำคัญของดัชนีสี (color index) ในการแสดงผล ถ้าเป็นศูนย์ทุกสีจะมีความสำคัญเท่ากันหมด

* หมายเหตุความหมายของจำนวนบิทต่อจุดภาพ

1	*บิทแมพ 2 สี (monochrome bitmap)*	*มีจำนวนแอร์เรย์สี 2 แอร์เรย์*
4	*บิทแมพมี 16 สีสูงสุด*	*มีจำนวนแอร์เรย์สี 16 แอร์เรย์*
8	*บิทแมพมี 256 สีสูงสุด*	*มีจำนวนแอร์เรย์สี 256 แอร์เรย์*
24	*บิทแมพมี 16.7 ล้านสีสูงสุด*	*NULL (ไม่มีแอร์เรย์)*

** สำหรับรูปแบบการบีบอัด มีความหมายดังนี้คือ

BI_RGB	แสดงว่าข้อมูลไม่มีการบีบอัด
BI_RLE4	ทำการเข้ารหัสรันเลนส์ (run-length) ด้วยขนาด 4 บิทต่อจุดภาพ โดยในสองไบต์จะประกอบด้วยไบต์นับ (count byte) และดัชนีสีขนาด 1 ไบต์ที่แสดงผลได้ 2 จุดภาพ
BI_RLE8	ทำการเข้ารหัสรันเลนส์ด้วยขนาด 8 บิท ต่อจุดภาพ โดยในสองไบต์จะประกอบด้วยไบต์นับและดัชนีสีขนาด 1 ไบต์ ในการแสดงผล 1 จุดภาพ

*** จำนวนดัชนีสีในตารางสีมีความหมายดังนี้คือ

0	ใช้จำนวนสีสูงสุดเท่ากับ $2^{biBitCout}$
ไม่เป็นศูนย์	- ถ้า biBitCount<24 biClrUsed จะระบุจำนวนสีที่ใช้ในการแสดงผล เช่น เมื่อ biBitCount เท่ากับ 8 จะมีค่า biClrUsed เท่ากับ 256 สี เป็นต้น - ถ้า biBitCount = 24 biClrUsed จะระบุขนาดของตารางอ้างอิง (reference table) เพื่อให้วินโดว์ได้รับรู้และใช้ในการแสดงผลพาเลตส์สีให้เหมาะสมที่สุด

สำหรับโครงสร้าง RGBQUAD ที่เก็บรายละเอียดของพาเลตของสีประกอบด้วยส่วนประกอบดังนี้

BYTE rgbBlue;

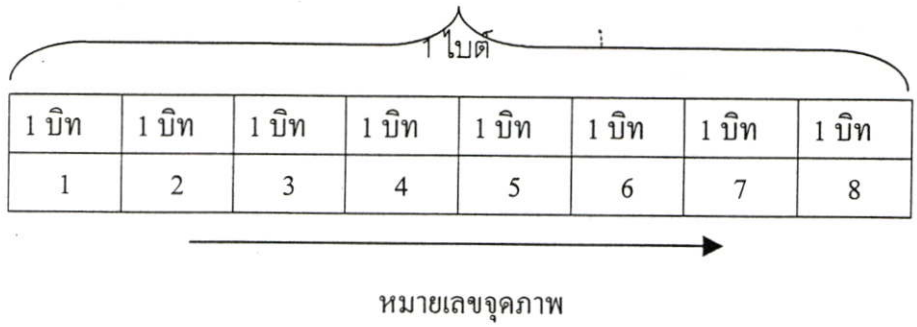
BYTE rgbGreen;

BYTE rgbRed;

BYTE rgbReserved; ไม่ใช้งาน (ปกติตั้งให้เป็นศูนย์)

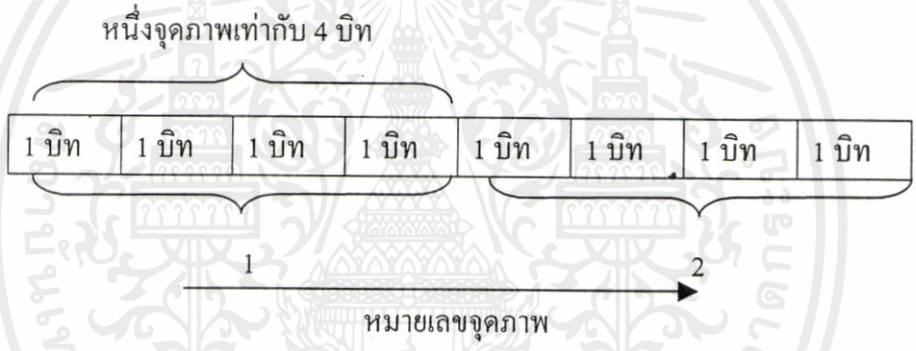
โดยทั่วไปข้อมูลบิทแมพของภาพจริงนั้นจะมีการเก็บที่แตกต่างกันตามจำนวนบิทข้อมูลภาพและลักษณะการบีบอัด แต่พบว่าข้อมูลในรูปแบบบิทแมพที่ใช้ส่งข้ามระหว่างโปรแกรมประยุกต์ต่าง ๆ นั้น จะไม่ใช้การบีบอัดข้อมูล สามารถแบ่งรูปแบบข้อมูลได้เป็น 4 ชนิด คือ

ชนิดที่ 1 ถ้าเป็น 1 บิทต่อจุดภาพ หมายความว่าดัชนีสีเป็นแอร์เรย์จำนวน 2 แอร์เรย์ คือมี 2 สี โดยข้อมูลแต่ละบิทจะแทนหนึ่งจุดภาพ ตามภาพที่ 2.8



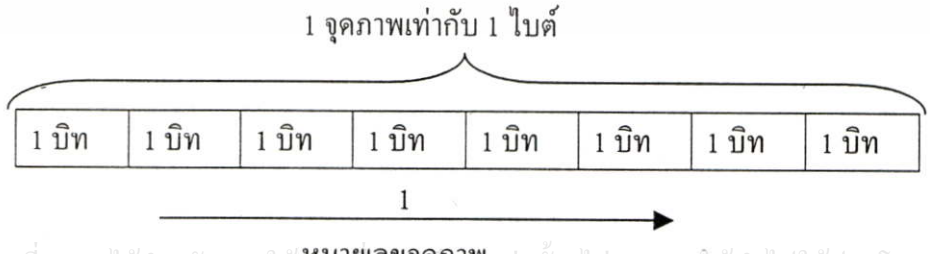
ภาพที่ 2.8 ข้อมูลภาพบิตแมพ 2 สี (monochrome Bitmap)

ชนิดที่ 2 ถ้าเป็น 4 บิทต่อจุดภาพ หมายความว่าดัชนีสีเป็นแอร์เรย์จำนวน 16 แอร์เรย์ คือมี 16 สี โดยข้อมูลแต่ละ 4 บิทจะแทนหนึ่งจุดภาพ ตามภาพที่ 2.9



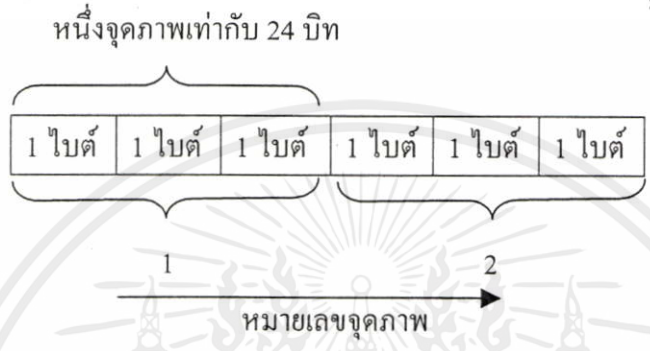
ภาพที่ 2.9 ข้อมูลภาพบิตแมพ 16 สี

ชนิดที่ 3 ถ้าเป็น 8 บิทต่อจุดภาพ หมายความว่าดัชนีสีเป็นแอร์เรย์จำนวน 256 แอร์เรย์ คือมี 256 สี โดยข้อมูลแต่ละ 8 บิทจะแทนหนึ่งจุดภาพ ตามภาพที่ 2.10



ภาพที่ 2.10 ข้อมูลภาพบิตแมพ 256 สี

ชนิดที่ 4 ถ้าเป็น 24 บิทต่อจุดภาพ หมายความว่าสามารถแสดงสีได้พร้อม ๆ กันเท่ากับ 2^{24} คือ 16.7 ล้านสี นั้นก็หมายความว่ารูปแบบของข้อมูลชนิดนี้ไม่ต้องมีการใช้ดัชนีสี เพราะสามารถที่จะผสมสีได้โดยตรง ดังนั้นในกรณีนี้ bmiColors = NULL ข้อมูลแต่ละ 3 ไบต์ที่เรียงกันจะแสดง relative intensity ของสีฟ้า สีเขียว และสีแดง (BRG) ตามลำดับ โดยข้อมูลแต่ละ 24 บิทจะแทนหนึ่งจุดภาพ ตามภาพที่ 2.11



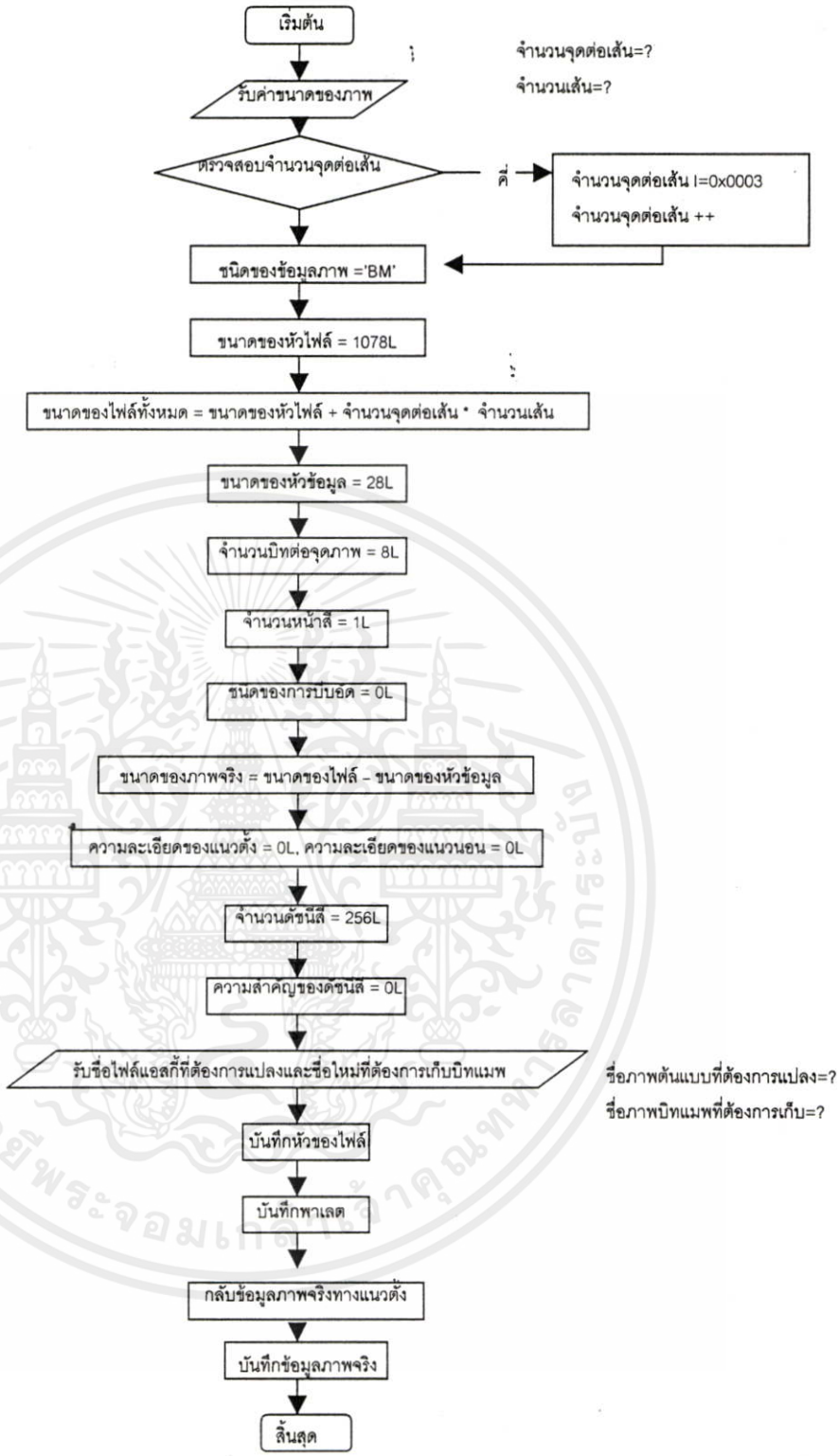
ภาพที่ 2.11 ข้อมูลภาพบิตแมพ 16.7 ล้านสี

2.6.2 การแปลงรูปแบบข้อมูลภาพ

เพื่อให้สามารถติดต่อสื่อสารกับโปรแกรมประยุกต์อื่น ๆ ได้นั้นจำเป็นต้องมีการแปลงภาพถ่ายแอสกี้ 8 บิท ให้เป็นภาพแบบบิทแมพที่มีหัวไฟล์เพิ่มเติมเข้ามาเพื่อบอกรายละเอียดต่าง ๆ ที่ให้ความสะดวกแก่การใช้งาน และในการแปลงกลับภาพแบบบิทแมพให้กลับมาเป็นให้กลับมาเป็นภาพถ่ายแอสกี้ 8 บิท

2.6.3 การแปลงภาพแอสกี้เป็นรูปแบบบิทแมพ

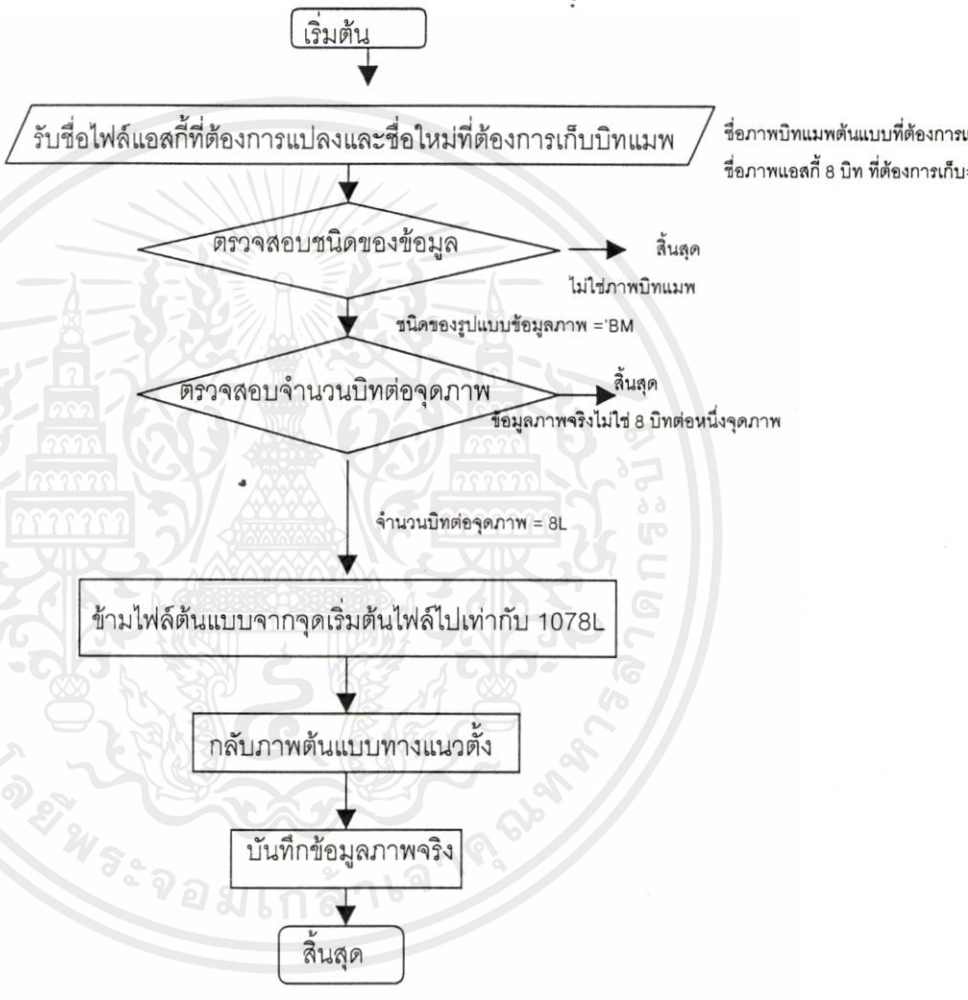
ภาพถ่ายที่ต้องการแปลงเป็นภาพสีเทาระหัสแอสกี้ขนาด 8 บิทต่อหนึ่งจุดภาพ เป็นข้อมูลภาพทั้งหมดโดยไม่มีเฮดเดอร์ ในการแปลงจะใส่เฮดเดอร์ให้โดยจะรับขนาดจุดภาพต่อหนึ่งเส้น (ขนาดทางแกน x) จำนวนเส้นของภาพ (ขนาดทางแกน y) และรับชื่อไฟล์ที่ต้องการแปลง โดยมีลำดับอัลกอริทึมในการทำงานของโปรแกรมดังนี้



ภาพที่ 2.12 แผนผังขั้นตอนการแปลงข้อมูลภาพแอสกี้ 8 บิทเป็นรูปบิทแมพ

2.6.4 การแปลงภาพในรูปแบบบิทแมพเป็นภาพแอสกี้

ภาพถ่ายที่ต้องการแปลงเป็นภาพสีเทาในรูปแบบบิทแมพขนาด 8 บิทต่อหนึ่งจุดภาพเป็นข้อมูลที่มีเฮดเดอร์ ในการแปลงจะปลดเฮดเดอร์ออกโดยจะรับไฟล์ที่ต้องการแปลงเท่านั้น สำหรับขนาดจุดภาพต่อหนึ่งเส้น (ขนาดทางแกน x) จำนวนเส้นของภาพ (ขนาดทางแกน y) โปรแกรมจะอ่านออกมาจากเฮดเดอร์ ลำดับอัลกอริทึมในการทำงานของโปรแกรมดังนี้



ภาพที่ 2.13 แผนผังขั้นตอนการแปลงภาพในรูปแบบบิทแมพเป็นข้อมูลภาพแอสกี้ 8 บิท

บทที่ 3
การกรองมัธยฐาน

การกรองมัธยฐาน (Median filter) จัดเป็นประเภทตัวกรองแบบไม่เป็นเชิงเส้น (Nonlinear filter) โดยอยู่บนพื้นฐานของการจัดอันดับทางสถิติ (Based on order statistics) ในต้นทศวรรษ 1970 Tukey [1] ได้นำเสนอการกรองมัธยฐานเป็นครั้งแรก ข้อดีของการกรองมัธยฐานคือสามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์ได้ดี และมีคุณสมบัติในการรักษาขอบภาพได้ด้วย เปรียบเทียบกับการกรองแบบเชิงเส้น (Linear filter) ประเภท Linear averaging การกรองมัธยฐานมีข้อดีคือใช้ในการคำนวณได้ง่ายและสามารถปรับเปลี่ยนหรือประยุกต์ไปใช้ในงานอื่น ๆ ได้มากมาย เช่น การประมวลผลทางภาพ [3,4], digital image analysis [5,6], digital TV application [7,8], การประมวลผลทางเสียงและเข้ารหัส [9,10]

3.1 ความหมายของการกรองมัธยฐาน

การกรองมัธยฐาน หมายถึง การหาข้อมูลของสมาชิก ณ จุดกึ่งกลาง อธิบายได้ตามสมการที่ 3.1

โดยกำหนดให้ n คือจำนวนตัวอย่างที่จะหาค่ามัธยฐาน

x_i คือข้อมูลที่ตำแหน่งต่าง ๆ เมื่อ i =1,2,3,..., n จะได้ว่า

$$med(x_i)=\begin{cases} x_{(v+1)} & ,n=2v+1 \\ \frac{1}{2}\left(x_{(v)}+x_{(v+1)}\right) & ,n=2v \end{cases} \qquad (3.1)$$

โดยที่ x_i คือข้อมูลที่มีการจัดอันดับทางสถิติแล้ว สมการ 3.1 จะมีความเหมาะสมกับขนาดของ n ที่เป็นเลขคี่อย่างมาก

ในการกรองมัธยฐานแบบ 1 มิติ อธิบายได้ตามสมการที่ 3.2 ค่า $n=2v+1$ คือขนาดของ filter window ที่จะใช้นั่นเอง

$$y_i = med\left(x_{i-v},\ldots,x_i,\ldots,x_{i+v}\right) \qquad i \in Z \qquad (3.2)$$

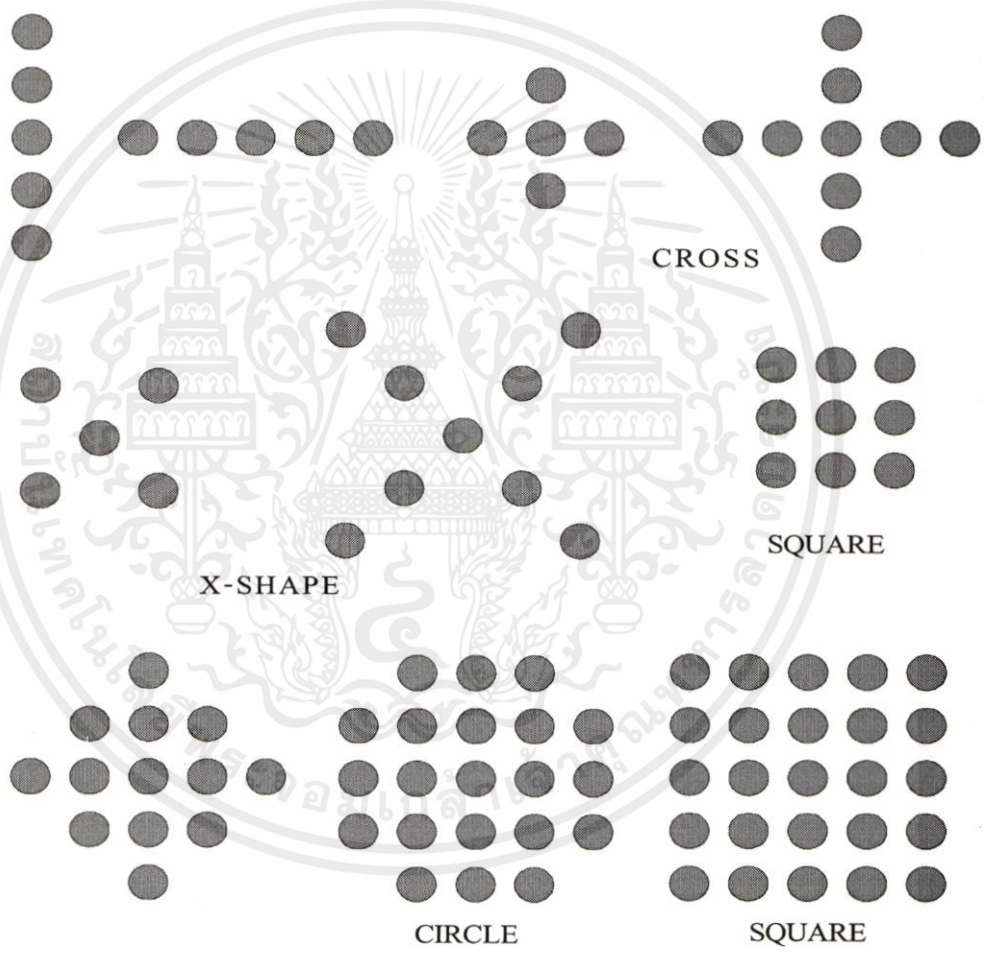
x_i คืออินพุท

Z คือ เลขจำนวนเต็ม

สำหรับสมการการกรองมัธยฐาน 2 มิติ อธิบายได้ตามสมการที่ 3.3

$$y_{ij} = med\left(x_{i+r,j+s}\ ;\ (r,s) \in A\right) \qquad i,j \in Z^2 \qquad (3.3)$$

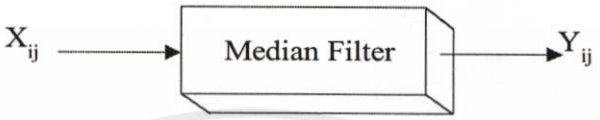
เมื่อ $A \in Z^2$ เป็นการกำหนดหน้าต่าง (filter window) โดยมีพิกเซลกึ่งกลางที่ตำแหน่ง (i,j) รูปแบบของขนาดหน้าต่างที่นิยมใช้สำหรับการกรองมัธยฐานแสดงในภาพที่ 3.1



ภาพที่ 3.1 รูปแบบหน้าต่าง (Filter windows) แบบต่างๆ สำหรับการกรองมัธยฐาน (Median filter)

รูปแบบหน้าต่างข้างต้นจะประกอบด้วยพิกเซลที่อยู่รอบจุดศูนย์กลางคือจุดกึ่งกลางและพิเซลในตำแหน่งกึ่งกลาง (central pixel)

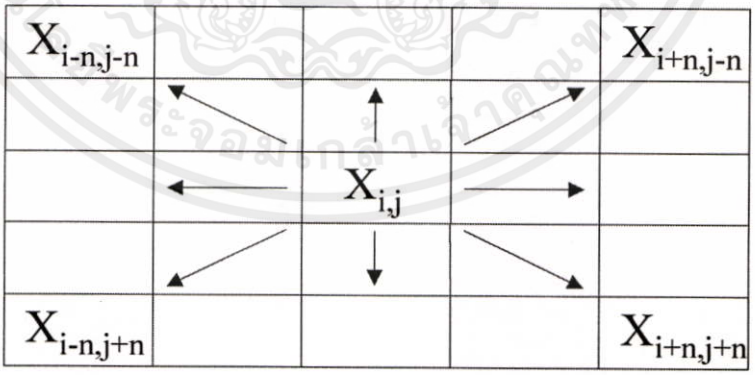
รูปแบบหน้าต่างข้างต้นจะเป็นทั้งแบบ 1 และ 2 มิติ ซึ่งได้ถูกดัดแปลงไปใช้ในงานประเภทต่าง ๆ กันขึ้นอยู่กับความเหมาะสม นอกจากรูปแบบที่ยกขึ้นมาแล้วยังอาจมีรูปแบบอื่น ๆ อีกมากมายที่ได้มีนักวิจัยท่านอื่น ๆ ได้คิดค้นขึ้น การกรองมัธยฐานสามารถแสดงเป็นภาพเพื่อความเข้าใจที่ดีขึ้น (ตามภาพที่ 3.2)



ภาพที่ 3.2 การกรองมัธยฐาน

3.1.1 การกรองมัธยฐานแบบหน้าต่างคงที่

การหาค่ามัธยฐานทำได้โดยการนำข้อมูลทั้งหมดในหน้าต่างรูปแบบที่เราเลือกใช้มาทำการเรียงค่าใหม่ ตามอันดับทางสถิติจากต่ำไปหาสูงหรือจากสูงมาหาต่ำก็ได้ เราจะพบว่าข้อมูลได้ถูกแบ่งเป็น 3 กลุ่ม คือ สมาชิกจำนวนครึ่งหนึ่งมีค่าต่ำกว่าค่าที่จุดกึ่งกลาง สมาชิกในตำแหน่งกึ่งกลาง และสามาชิกอีกครึ่งหนึ่งในที่มีค่าสูงกว่าค่าที่จุดกึ่งกลาง ค่ามัธยฐาน ก็คือการหาค่าข้อมูลของสมาชิก ณ จุดกึ่งกลาง (ภาพที่ 3.3 และ 3.4)



ภาพที่ 3.3 หน้าต่างขนาด NxN บนภาพ $X_{i,j}$ ที่จุด i, j

ให้ $W_{i,j}$ เป็นเซ็ตของจุดภาพในหน้าต่างขนาด NxN (โดยที่ $N = 2n+1$) ที่จุด $X_{i,j}$ สามารถแสดงได้ตามสมการ

$$W_{i,j} = \{X_{i-n,j-n},...,X_{i-n,j-n+1},...,X_{i,j},...,X_{i+n,j+n}\} \quad (3.4)$$

นำมาจัดลำดับทางขนาดหรืออันดับทางสถิติ $W_1^{i,j}$ ของเซตข้อมูล $W_{i,j}$ ตามเงื่อนไขได้ตามสมการที่ 3.5

$$W_1^{i,j} \le W_2^{i,j} \le,...,\le W_{NxN}^{i,j} \quad (3.5)$$

โดยที่ K คือ 1,2,...,NxN เราจะได้ว่า $W_1^{i,j}$ คือค่าอันดับทางสถิติต่ำสุด $W_{(NxN+1)/2}^{i,j}$ คือค่าอันดับทางสถิติกลาง $W_{NxN}^{i,j}$ คือค่าอันดับสถิติสูงสุด ค่ามัธยฐานคือค่าอันดับทางสถิติกลางหรือสามารถกำหนดเป็นสมการที่ 3.6 และ 3.7 ตามลำดับ

$$Y_{i,j} = med[W_{i,j}] \quad (3.6)$$

$$= W_{(NxN+1)/2}^{i,j} \quad (3.7)$$

3.1.2 การกรองค่าเฉลี่ย (Average Filter)

การกรองค่าเฉลี่ยเป็นการประมวลผลสัญญาณเชิงเส้น (Linear Signal Processing) มีคุณสมบัติเป็นตัวกรองความถี่ต่ำ (Lowpass filter) ทำให้ภาพมีความราบเรียบขึ้น น้ำหนักของภาพจะใกล้เคียงกัน โดยใช้ค่าเฉลี่ยในการประมาณค่าของข้อมูลภาพใหม่ $\hat{S}_{i,j}$ จากข้อมูลภาพในขนาดหน้าต่าง NxN บนภาพ $Y_{i,j}$ ที่จุด i, j โดยมีลักษณะทิศทางการวิ่งของหน้าต่างเหมือนกับรูปการกรองมัธยฐาน สมการของการกรองค่าเฉลี่ยเขียนได้ตามสมการที่ 3.8

$$\hat{S}_{i,j} = \frac{1}{NxN} \sum_{k=-n}^{n} \sum_{l=-n}^{n} Y_{i-k,j-l} \quad (3.8)$$

เมื่อ $N = 2n+1$

ผลของการกำจัดสัญญาณรบกวนแบบอิมพัลส์โดยใช้การกรองค่าเฉลี่ยถูกแสดงในภาพที่ 3.5

(ก)

(ข)

(ค)

(ง)

ภาพที่ 3.4 ภาพผลลัพธ์จากภาพ Lena ขนาด 256x256 ต้นแบบ (ก)

ภาพ Lena ที่ถูกรบกวนโดยสัญญาณรบกวนแบบ salt and pepper noise (impulse) 5% (ข)

การกรองมัธยฐานแบบหน้าต่างคงที่ขนาด 3x3 (ค)

การกรองมัธยฐานแบบหน้าต่างคงที่ขนาด 5x5 (ง)

(ก)

(ข)

(ค)

(ง)

ภาพที่ 3.5 ภาพ Lena ที่มีสัญญาณรบกวนแบบ salt and pepper noise (impulse) 5% (ก)
การกรองค่าเฉลี่ยแบบหน้าต่างคงที่ขนาด 3x3 (ข)
กรองค่าเฉลี่ยแบบหน้าต่างคงที่ขนาด 5x5 (ค)
กรองค่าเฉลี่ยแบบหน้าต่างคงที่ขนาด 7x7 (ง)

3.2 ความรู้ทั่วไปในการปรับปรุงภาพ

3.2.1 การทำคอนโวลูชันในสปาเชียลโดเมน

การทำคอนโวลูชันเป็นการทำงานในรูปแบบที่เรียกว่า neighborhood operation ซึ่งเป็นการทำงานที่ต้องใช้จุดข้างเคียงมาพิจารณาในการคำนวณ การทำ neighborhood operation นั้นผลลัพธ์ได้ค่าออกมา ณ ตำแหน่งเดียวกันกับอินพุตและใช้จุดข้างเคียงที่อยู่รอบ ๆ ซึ่งการทำคอนโวลูชันเป็นการคำนวณผลรวมของผลคูณ (sum of product) แสดงหน้ากากของคอนโวลูชัน (convolution mask) ตามภาพที่ 3.6 และจุดข้างเคียงจากภาพตามภาพที่ 3.8 สมาชิกแต่ละตัวในหน้ากากเรียกว่าน้ำหนัก (weight) ค่าน้ำหนักในหน้ากากเป็นตัวคำนวณผลของการทำคอนโวลูชัน ซึ่งกำหนดลักษณะการกรองตามการประยุกต์ใช้ในงานต่าง ๆ

$M(-1,-1)$	$M(0,-1)$	$M(1,-1)$
$M(-1,0)$	$M(0,0)$	$M(1,0)$
$M(-1,1)$	$M(0,1)$	$M(1,1)$

ภาพที่ 3.6 หน้ากากของการทำคอนโวลูชันขนาด 3x3

...	...	...	...	...
...	$F(i-1),j-1)$	$F(i,j-1)$	$F(i+1,j-1)$	...
...	$F(i-1,j\quad)$	$F(i,j\quad)$	$F(i+1,j)$	...
...	$F(i-1,j+1)$	$F(i,j+1)$	$F(i+1,j+1)$	...
...	...	...	...	...

ภาพที่ 3.7 จุดข้างเคียง (neighborhood) ต่าง ๆ ในบริเวณเมตริกซ์ขนาด 3x3 ของภาพต้นแบบ

ตัวชี้ในหน้ากากมีขุดเริ่มต้นที่จุดศูนย์กลาง โดยเริ่มที่ตำแหน่งมุมบนด้านซ้ายของภาพ สามารถเขียนสมการคอนโวลูชันตามสมการที่ 3.9

$$C(i,j)=\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1}F(k,l)M(i-k,j-l) \qquad (3.9)$$

แต่ละจุดภาพจะเป็นการวนเอามาจากจุดรอบข้างและคูรด้วยค่าที่ตรงตำแหน่งเดียวกันกับหน้ากาก นำผลคูณของแต่ละตัวนั้นมารวมกันจะได้ค่าจุดภาพผลลัพธ์ ซึ่งการคำนวณที่ทำนั้นสามารถกระจายแสดงออกมาได้ตาม 3.10

$$\begin{aligned} C(i,j) = {} & F(i-1,j-1)M(1,1) + F(i,j-1)M(0,1) + F(i+1,j-1)M(-1,1) + \\ & F(i-1,j\ \ )M(1,0) + F(i,j\ \ )M(0,0) + F(i+1,j\ \ )M(-1,0) + \\ & F(i-1,j+1)M(1,-1) + F(i,j+1)M(0,-1) + F(i+1,j+1)M(-1,-1) \end{aligned} \tag{3.10}$$

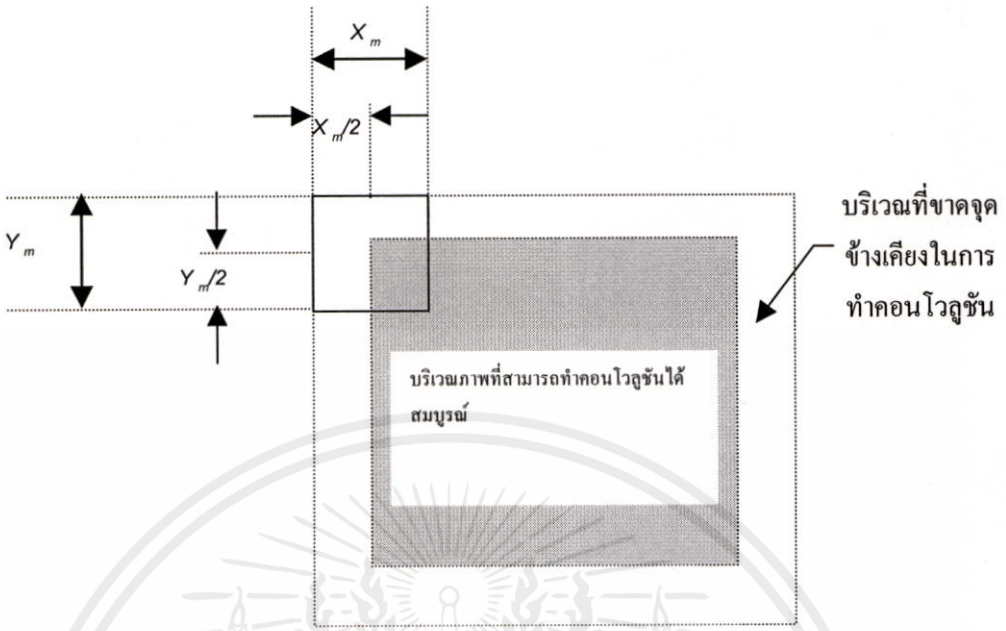
จากสมการที่ 3.10 แสดงการจัดเรียงเทอมต่าง ๆ โดยลำดับหน้ากากมีลำดับจากบนซ้ายไปยังด้านล่างขวา สำหรับค่าตำแหน่งที่ตรงกันในหน้ากากจะมีลำดับตรงข้ามคือเริ่มจากด้านล่างขวาขึ้นไปยังด้านบนซ้าย สำหรับสิ่งที่น่าสนใจของการตรงกันข้ามของลำดับนี้ก็คือว่าถ้าหมุนหน้ากาก 180^0 ลำดับของเทอมภาพกับเอมหน้ากากจะมีลำดับในทิศทางเดียวกันโดยได้การกระจายตาม 3.11

$$\begin{aligned} C(i,j) = {} & F(i-1,j-1)M(-1,-1) + F(i,j-1)M(0,-1) + F(i+1,j-1)M(1,-1) + \\ & F(i-1,j\ \ )M(-1,\ 0) + F(i,j\ \ )M(0,\ 0) + F(i+1,j\ \ )M(1,\ 0) + \\ & F(i-1,j+1)M(-1,-1) + F(i,j+1)M(0,1) + F(i+1,j+1\ )M(1,\ 1) \end{aligned} \tag{3.11}$$

ในการประยุกต์ใช้งานบางอย่างนิยมใช้การหมุนหน้ากาก 180^0 นี้เพื่อให้ง่ายต่อการกำหนดตัวชี้ให้ภาพและหน้ากากมีตัวชี้ที่เหมือนกัน ในการประยุกต์บางอย่างจะมีการจัดเรียงสมาชิกของหน้ากากอย่างอัตโนมัติ โดยมีตำแหน่งของหน้ากากและภาพที่สอดคล้องกัน อีกทั้งหน้ากากก็ไม่จำเป็นที่จะต้องมีจำนวนแนวตั้งเท่ากับจำนวนแนวนอนเสมอไปด้วย แต่โดยส่วนใหญ่แล้วหน้ากากมักจะสมมาตรกัน (symmetric) และโดยทั่วไปความกว้างและความสูงจะเป็นจำนวนคี่เพื่อให้จุดศูนย์กลางของหน้ากากเป็นจุดศูนย์กลางได้

3.2.2 การขาดจุดข้างเคียงในบริเวณของการทำคอนโวลูชันในบริเวณกรอบภาพ

ในขบวนการคำนวณแบบ neighborhood operation นี้เป็นไปไม่ได้ที่จะทำคอนโวลูชันในบริเวณขอบรอบนอกของภาพ (บริเวณกรอบภาพ) เพราะในการคำนวณขาดจุดภาพที่อยู่ภายนอกอาณาบริเวณของภาพ (ภาพที่ 3.8)



ภาพที่ 3.8 การขาดจุดข้างเคียงในบริเวณของการทำคอนโวลูชันในบริเวณกรอบภาพ

โดยทั่วไปแล้ววิธีการที่จะแก้ปัญหาเหล่านี้ได้ก็มักใช้วิธีการกำหนดแถบจุดภาพในบริเวณกรอบของภาพทึ้ง 4 ด้าน ให้เป็นค่าเดิมหรือเป็นศูนย์แล้วแต่กรณี ความกว้างของแถบด้านบนและด้านล่างของภาพมีค่าเท่ากับ Ym/2 ส่วนด้านซ้ายและขวาเท่ากับ Xm/2 เมื่อ Xm คือความกว้างของหน้ากาก และ Ym คือความสูงของหน้ากากตัวอย่าง เช่น หน้ากากขนาด 3x3 จะมีแถบที่มีความกว้าง 1 จุดภาพ ซึ่งไม่สามารถคำนวณรอบ ๆ ภาพได้

ในการใช้งานจะต้องรู้ว่าจะนำภาพที่ได้ไปทำอะไรต่อไป เพราะอาจจะเป็นไปได้ที่ภาพผลลัพธ์อาจจะมีขนาดเล็กกว่าภาพต้นแบบตามจำนวนโรว์และคอลัมน์ที่ไม่สามารถทำการคำนวณได้ ถ้าเงื่อนไขไม่เป็นเช่นนั้นและต้องการรักษาขนาดของภาพผลลัพธ์ให้เท่ากับภาพต้นแบบก็ใช้การใส่ค่า
เดิมหรือศูนย์แล้วแต่กรณีลงไปในกรอบที่ไม่สามารถคำนวณได้ ซึ่งวิธีนี้เป็นที่นิยมกันอย่างมาก เพราะภาพผลลัพธ์สามารถนำไปเปรียบเทียบกับภาพต้นแบบได้โดยตรงและในอนาคตก็สามารถนำไปคำนวณกันได้ต่อไปในขณะที่ยังรักษาแนวของจุดภาพให้ตรงกันอยู่อย่างนั้น

บางระบบต้องการการคำนวณที่สมบูรณ์แบบอย่างแท้จริง โดยการใช้ข้อมูลในบริเวณอื่น ๆ มาใช้แทนบริเวณโลและคอลัมน์ของจุดภาพที่อยู่เลยออกไปจากภาพต้นแบบ วิธีการหนึ่งนั้นทำได้โดยการใช้โรว์ของทางด้านล่างของภาพมาใช้เป็นข้อมูลที่ต้องการของการคำนวณทางด้านบนและ

ในทำนองเดียวกันโดยการใช้โรว์ของทางด้านบนของภาพมาใช้เป็นข้อมูลที่ต้องการของการคำนวณทางด้านล่าง ในลักษณะที่คล้ายกันของคอลัมน์ทางด้านขวาของภาพก็นำมาใช้เป็นข้อมูลที่ต้องการของการคำนวณทางด้านซ้ายเสร็จสมบูรณ์และคอลัมน์ทางด้านซ้ายของภาพก็นำมาใช้เป็นข้อมูลที่ต้องการของการคำนวณทางด้านขวาเช่นกัน ถึงแม้ว่าวิธีการเช่นนี้ไม่เป็นการเหมาะสมนักก็ตาม แต่เพื่อให้ผลลัพธ์มีขนาดภาพเท่ากับภาพต้นแบบและในการประยุกต์ใช้งานบางอย่างต้องหลีกเลี่ยงการเกิดการเพี้ยนอย่างรุนแรงของวิธีการคำนวณจากการแทนกรอบที่คำนวณไม่ได้ด้วยศูนย์เช่นในกรณีทำเกรเดียนท์ เป็นต้น

3.2.3 การนอร์มอลไลซ์ (Normalization)

ในบางกรณีที่ต้องการทำการประมวลผลให้ข้อมูลมีความราบเรียบ (smoothing) เพื่อกำจัดสัญญาณรบกวนนั้น คุณสมบัติบางประการของข้อมูลภาพเดิมจำเป็นจะต้องรักษาเอาไว้ ในที่นี้คือคุณสมบัติเอกพันธ์หรือความเป็นเนื้อเดียวกันของกลุ่มภาพข้างเคียง ถ้าหากพื้นที่ใด ๆ มีความเป็นเนื้อเดียวกันอยู่แล้ว หลังจากการประมวลบผลในการทำข้อมูลให้ราบเรียงจะต้องยังคงไว้ซึ่งความเป็นเนื้อเดียวเช่นเดิม ดังนั้นหน้ากากที่ใช้ในการประมวลผลความราบเรียบของข้อมูลจึงจำเป็นต้องมีการนอร์มอลไลซ์ ของค่าที่ใช้นอร์มอลไลซ์ในการณีนี้คือผลรวมของน้ำหนักแต่ละจุดในหน้ากากนั้นเอง

ตัวอย่าง สมมุติค่าน้ำหนักของหน้ากากอันหนึ่งเป็น{1,2,1,2,4,2,1,2,1} (เมื่อการรวมค่าน้ำหนักเราจะได้ 1+2+1+2+4+2+1+2+1 = 16 ก็ทำการนอร์มอลไลซ์ด้วย 16 ตามสมการที่ 3.12

$$M(i,j) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad (3.12)$$

3.2.4 เวลาที่ใช้

การทำคอนโวลูชันค่อนข้างจะใช้การคำนวณมากครั้ง เช่น ถ้าเป็นหน้ากาก 3x3 ในการที่จะได้ผลลัพธ์ 1 จุดภาพจะต้องใช้การคูณ 9 ครั้ง การบวก 9 ครั้ง และหารหารอีก 1 ครั้ง ถ้ามีการทำนอร์มอลไลซ์พิจารณาการทำคอนโลูชันของหน้ากากที่มีโรว์เท่ากับคอลัมน์แล้วจำนวนการบวกจะท่ากับจำนวนการคูณ ตารางที่ 3.1 แสดงจำนวนครั้งที่ต้องใช้ในการคำนวณ โดยทดลองใช้หน้ากากขนาดต่าง ๆ โดยใช้ภาพต้นแบบขนาด 512x512 โดยสมมุติให้สามารถทำคอนโวลูชันที่บริเวณกรอบของภาพได้ เช่น ใช้หน้ากากขนาด 3x3 ต้องใช้จำนวนครั้งในการคำนวณเท่ากับ 2,359,296x2+262,144 = 4,980,736 ครั้ง เป็นต้น

ตารางที่ 3.1 การคำนวณจากตารางหน้ากากจตุรัสขนาดต่าง ๆ กับภาพต้นแบบขนาด 512x512 จุดภาพ

ขนาดหน้าต่าง	จำนวนสมาชิกในหน้ากาก	จำนวนครั้งในการบวก และคูณ	จำนวนครั้งในการหาร	รวมจำนวนครั้งทั้งหมด
3	9	2,359,296	262,144	4,980,736
5	25	6,553,600	262,144	13,369,344
7	49	12,845,056	262,144	25,952,256
9	81	21,233,664	262,144	42,729,472
11	121	31,719,424	262,144	63,700,992
15	225	58,982,400	262,144	118,226,944
25	625	163,840,000	262,144	327,942,144

เมื่อขนาดของหน้ากากใหญ่ขึ้นจะเห็นว่าจำนวนครั้งในการใช้คำนวณจะเพิ่มขึ้นมากกอย่างมาก ซึ่งในตารางนี้จะเป็นการคิดจำนวนครั้งในการคำนวณเท่านั้นโดยไม่ได้นับการทำงานที่แฝงอยู่อื่น ๆ ที่จำเป็นต้องใช้เลย ได้แก่ การแยกจุดภาพที่ถูกต้องจากภาพและหน้ากาก เขียนเก็บข้อมูลของภาพผลลัพธ์รักษาดัชนีตัวชี้และตัวนับและควบคุมการทำงานทั้งหมด ดังนั้นจำนวนครั้งในการทำงานทั้งหมดในโครงสร้างการประมลผลจึงใช้เวลามากกว่าที่แสดงมาในตารางข้างต้นมาก ดังนั้นเมื่อต้องมีการทำคอนโวลูชันจึงมักนิยมใช้ขนาดของหน้ากากที่เล็กที่สุดเท่าที่จะทำได้

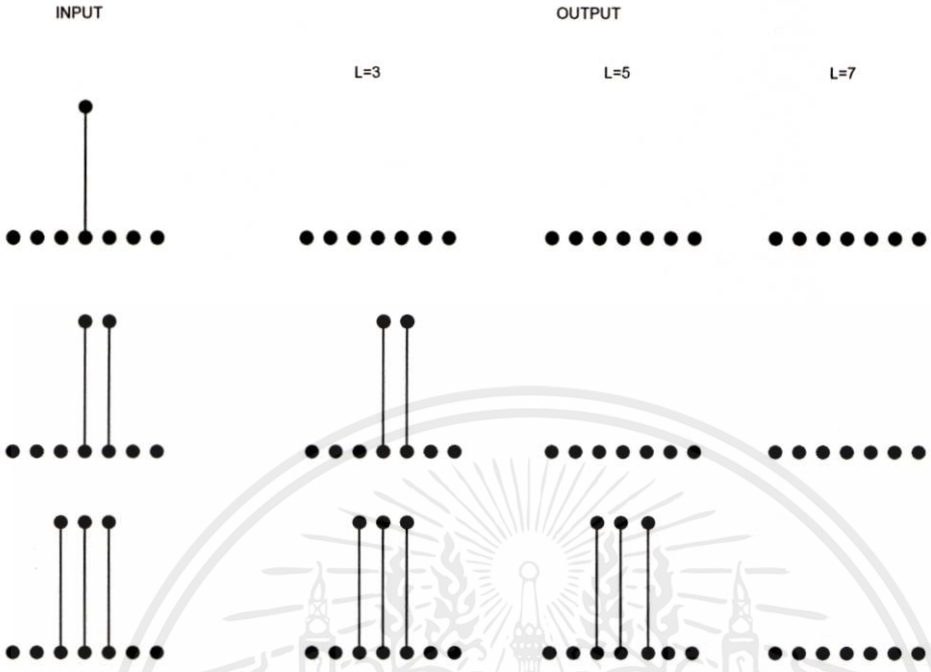
บทที่ 4
การกรองมัธยฐานแบบปรับขนาดอัตโนมัติ

จากพื้นฐานของการกรองมัธยฐานนั้น จัดเป็นเทคนิคอย่างหนึ่งของการประมวลผลสัญญาณแบบไม่เป็นเชิงเส้น เราทราบกันดีอยู่แล้วว่า จะสามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์นอยส์ได้ดี แต่ถ้าใช้กับหน้าต่างขนาดใหญ่จะทำให้ภาพผลลัพธ์มีความเบลอสูง ดังดูได้จากภาพตัวอย่างในบทที่ 3 เช่น การกรองมัธยฐานแบบ 5x5 หรือ 7x7 แต่ถ้าใช้กับหน้าต่างขนาดเล็ก ก็อาจทำให้กำจัดสัญญาณรบกวนได้ไม่หมดในกรณีมีสัญญาณรบกวนแบบอิมพัลส์เป็นจำนวนมาก บทนี้จะนำเสนอเทคนิคและวิธีการในการประยุกต์ใช้งาน การกรองมัธยฐานให้มีประสิทธิภาพมากขึ้น โดยจะเป็นการปรับลดหรือเพิ่มขนาดหน้าต่างของการกรองมัธยฐานอัตโนมัติ ขึ้นอยู่กับปริมาณของสัญญาณรบกวนแบบอิมพัลส์ โดยเงื่อนไขการปรับ จะขึ้นอยู่กับความแตกต่างของระดับเทรชโฮลด์ของภาพระดับเทา (Gray scale level image) ที่เรากำหนด

หลักการทำงานของการกรองมัธยฐานแบบปรับขนาดหน้าต่างได้อย่างอัตโนมัติ (Adaptive window size median filter)

4.1 แนวความคิดแบบ 1 มิติ

จากคุณสมบัติของการกรองมัธยฐานและค่าทางสถิติ เราจะสามารถอธิบายภาพที่ 4.1 ได้ดังนี้แถวซ้ายสุด (Input signal) จะเป็นภาพที่ประกอบไปด้วยสัญญาณรบกวนที่เข้ามาในบริเวณหน้าต่าง (อิมพัลส์นอยส์ ถูกแสดงโดยจุดที่สูงกว่าจุดอื่น ๆ) ทางฝั่งขวาจะเป็นเอาท์พุทของการกรองมัธยฐานที่ขนาดหน้าต่างเริ่มตั้งแต่ความยาวเท่ากับ 3, 5 และ 7 จากทฤษฎีของการกรองมัธยฐานเราจะได้ว่า ถ้าอินพุทเป็นสัญญาณรบกวนขนาด 1 พิกเซล ขนาดหน้าต่างทั้ง 3 แบบ (L=3, L=5 และ L=7) จะสามารถ กำจัดสัญญาณรบกวนออกไปได้ ถ้าอินพุทเป็นสัญญาณรบกวนขนาด 2 พิกเซล หน้าต่างขนาด L=3 จะไม่สามารถกำจัดสัญญาณรบกวนออกไปได้ แต่หน้าต่างขนาด L=5 และ L=7 กำจัดได้ ถ้าอินพุทเป็นสัญญาณรบกวนขนาด 3 พิกเซล หน้าต่าง L=7 จะสามารถกำจัดสัญญาณรบกวนได้ แต่ L=3 และ 5 ไม่สามารถกำจัดออกไปได้ ดังนั้นแนวความคิดของวิธีการนี้คือจะต้องจำแนกให้ได้ว่ามีจำนวนอิมพัลส์นอยส์ในหน้าต่างขณะนั้นมีมากหรือน้อยเท่าไหร่ โดยหลักการทางอัลกอริทึมที่จะอธิบายดังต่อไปนี้



ภาพที่ 4.1 อิมพัลส์นอยส์และความสามารถในการกำจัดนอยส์ของขนาดหน้าต่าง (Filter windows) ที่ต่างกัน

กำหนดความยาวหน้าต่างกว้างที่สุด L=7
กำหนดตำแหน่งจุดภาพในหน้าต่างดังนี้

$$x(n-3), x(n-2), x(n-1), x(n), x(n+1), x(n+2), x(n+3) \qquad (4.1)$$

ให้ $P_j = x(n+j-1) - x(n+j)$ เมื่อ $j = 1,2,3$

และ $P_j = x(n+j+1) - x(n+j)$ เมื่อ $j = -1,-2,-3$ (4.2)

เราสามารถเลือกขนาดของหน้าต่างที่จะใช้ได้ดังต่อไปนี้

ถ้า ($P_2 > T_3$ และ $P_{-2} > T_3$) หรือ ($P_1 > T_3$ และ $P_{-3} > T_3$) หรือ
($P_3 > T_3$ และ $P_{-1} > T_3$) จะเลือกใช้หน้าต่างขนาด L=7

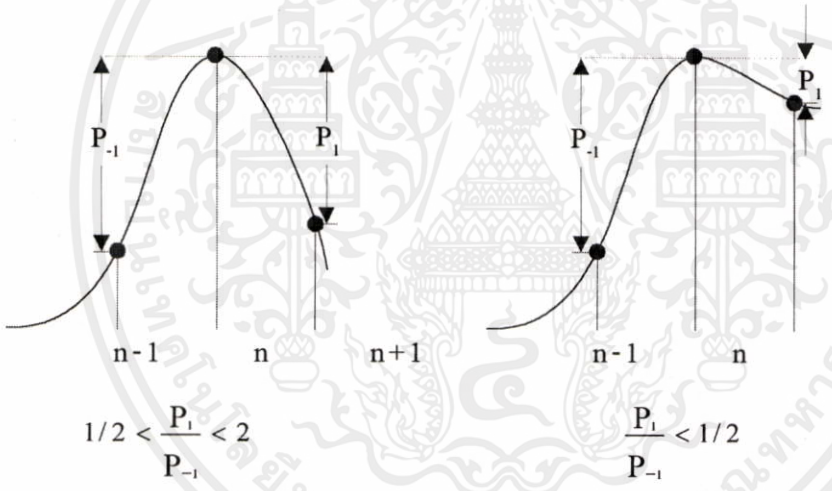
ถ้า ($P_1 > T_2$ และ $P_{-2} > T_2$) หรือ ($P_2 > T_2$ และ $P_{-1} > T_2$) จะเลือกใช้หน้าต่างขนาด L=5

ถ้า ($P_1 > T_1$ และ $P_{-1} > T_1$) จะเลือกใช้หน้าต่างขนาด L=3 (4.3)

ถ้าไม่ตรงกับทั้ง 3 เงื่อนไขข้างบนจะเลือก L=1 หรือค่าเดิมของจุดกึ่งกลางที่กำลังประมวลผล

ค่าเทรชโฮลด์ T_1, T_2, T_3 ที่ได้จากการทดลองซ้ำ ๆ หลาย ๆ ครั้ง จะอยู่ในช่วง 16 ถึง 48 (สำหรับระบบภาพแบบ 8 บิท/1 จุดภาพ) ซึ่งจะให้ผลลัพธ์ที่ไม่ค่อยแตกต่างกัน โดยที่ภาพผลลัพธ์ยังคงมีคุณภาพดีกว่าการกรองมัธยฐานแบบปกติ

สำหรับการแจกแจงของอิมพัลซ์นอยส์แบบสม่ำเสมอหรือการแจกแจงลอการิธึมแบบปกติ (Lognormal-distribution) อาจจะยังคงมีสัญญาณรบกวนแบบค่าต่ำ ๆ (Low-amplitude noise) อยู่บ้างเล็กน้อย เราจะเพิ่มเงื่อนไขเข้าไปอีก 1 ประการ เพื่อให้การกรองสัญญาณมีความสมบูรณ์มากยิ่งขึ้น เราคือใช้การตรวจสอบจับอิมพัลส์นอยส์แบบแอมปลิจูดต่ำ อธิบายได้ตามภาพที่ 4.2



ภาพที่ 4.2 อัตราส่วนของการตรวจจับอิมพัลส์นอยส์แบบแอมปลิจูดต่ำ

สำหรับ $P_1 < T_1$ หรือ $P_{-1} < T_1$

ถ้า $\frac{1}{2} < \frac{P_1}{P_{-1}} < 2$ เลือก L=3

ช่วงของค่า $\frac{P_1}{P_{-1}}$ ที่เราจะใช้ในการตรวจสอบจับอิมพัลซ์นอยส์แบบแอมปลิจูดต่ำถูกแสดงในภาพที่ 4.2 และเพื่อให้ง่ายต่อการนำไปประยุกต์ในการออกแบบวงจรรวมขนาดใหญ่ (VLSI:Very Large Scale Integrated Circuit) สามารถดูภาพผลลัพธ์ได้จากรูปที่ 4.3

(ก)

(ข)

(ค)

(ง)

ภาพที่ 4.3 ภาพ Lena ที่มีอิมพัลส์แบบบวก 5% (ก)
ภาพที่ผ่านการกรองมัธยฐานด้วยหน้าต่าง 1 มิติขนาด L=3 (ข)
ภาพที่ผ่านการกรองมัธยฐานด้วยหน้าต่าง 1 มิติ L=5 (ค)
ภาพที่การกรองมัธยฐานแบบปรับขนาดหน้าต่างได้แบบ 1 มิติ(ง)

ในกรณีที่สัญญาณรบกวนอิมพัลส์เป็นแบบลบ (Pepper noise or black dot) อัลกอริทึมดังกล่าวยังสามารถใช้ได้เพียงแต่เปลี่ยนเครื่องหมายของ P_j เป็นตรงกันข้ามแล้วดำเนินการตามขั้นตอนเหมือนกัน

การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติในรูป 1 มิติจะให้ผลลัพธ์ไม่ดีถ้าภาพนั้นมีสัญญาณรบกวนแบบอิมพัลส์ทั้ง 2 ชนิด คือทั้งบวกและลบหรือมีปริมาณสัญญาณรบกวนที่มาก เพราะว่าจะมีการผิดเพี้ยนของสัญญาณสูง อันเนื่องมาจากขนาดหน้าต่างที่เล็กเกินไป และการล้อมรอบของสมาชิกในฟิลเตอร์วินโดว์ ต่อพิกเซลที่จุดกึ่งกลางไม่เหมาะสม เนื่องจากสัญญาณภาพมีลักษณะเป็น 2 มิติ เพราะฉะนั้นการกรองมัธยฐานแบบ 2 มิติจึงมีความเหมาะสมมากกว่าในการเพิ่มความคมชัดและลดการผิดเพี้ยนของสัญญาณภาพ

4.2 การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ 2 มิติ

การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ 2 มิตินั้น แบ่งการทำงานออกเป็น 2 ส่วน สำหรับการกำจัดสัญญาณรบกวนอิมพัลส์แบบจุดขาว (Salt noise or white impulse) และการกำจัดสัญญาณรบกวนอิมพัลส์แบบจุดดำ (pepper noise or black impulse) โดยเราจะเลือกแบบของตารางหน้าต่างเป็นแบบกากบาท ขนาด 5x5 (ภาพที่ 4.4)

		y(n-2)		
		y(n-1)		
x(n-2)	x(n-1)	x(n) y(n)	x(n+1)	x(n+2)
		y(n+1)		
		y(n+2)		

ภาพที่ 4.4 การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ 2 มิติ

4.2.1 การกรองมัธยฐานแบบปรับขนาดได้อย่างอัตโนมัติที่ใช้กับ salt noise:

ตารางหน้าต่างเป็นแบบกากบาท

ขนาดสูงสุดของความยาวตารางหน้าต่างเท่ากับ 5x5

จุดในตารางหน้าต่างในทิศทางแนวนอน กำหนดโดย

$$x(n-2), x(n-1), x(n), x(n+1), x(n+2) \quad (4.4)$$

จุดในตารางหน้าต่างในทิศทางแนวตั้ง กำหนดโดย

$$y(n-2), y(n-1), y(n), y(n+1), y(n+2) \quad (4.5)$$

โดยที่ $x(n) = y(n)$ เท่ากับจุดกึ่งกลางของตารางหน้าต่าง

เรากำหนด เซ็ทของ P และ Q เป็นผลต่างของค่าระดับสีเทาของจุดภาพ ณ. ตำแหน่งต่างๆ ในตารางหน้าต่าง เพื่อเป็นการตรวจสอบความกว้างของสัญญาณรบกวนที่อยู่ในตารางหน้าต่าง และนำค่าเหล่านี้ไปใช้ในขบวนการตัดสินใจเพื่อเลือกขนาดของตารางหน้าต่าง

$$
\begin{aligned}
P_j &= x(n+j-1) - x(n+j) \quad \text{เมื่อ } j = 1, 2 ; \\
P_j &= x(n+j+1) - x(n+j) \quad \text{เมื่อ } j = -1, -2 ; \\
Q_j &= y(n+j-1) - y(n+j) \quad \text{เมื่อ } j = 1, 2 ; \\
Q_j &= y(n+j+1) - y(n+j) \quad \text{เมื่อ } j = -1, -2 ;
\end{aligned} \tag{4.6}
$$

กำหนดให้ L_h, L_v คือ ขนาดหน้าต่างในแนวนอนและแนวตั้งตามลำดับการพิจารณาเลือกขนาดของ L_h และ L_v ทำได้ดังนี้

I. การตรวจสอบทางแนวนอน :

ถ้า ($P_2 > T_2$ และ $P_{-2} > T_2$) หรือ ($P_1 > T_2$ และ $P_{-2} > T_2$) หรือ ($P_2 > T_2$ และ $P_{-1} > T_2$)

เลือก $L_h = 5$ แล้วไปตรวจสอบทางแนวตั้ง

ถ้า ($P_1 > T_1$ และ $P_{-1} > T_1$) หรือ ($1/2 < P_1/P_{-1} < 2$)

เลือก $L_h = 3$ แล้วไปตรวจสอบทางแนวตั้ง;

ถ้า ($-T_1 < P_1 < T_1$ และ $-T_1 < P_{-1} < T_1$ และ $-T_2 < P_2 < T_2$ และ $-T_2 < P_{-2} < T_2$)

เลือก $L_h = L_v = 1$ คือที่จุดกึ่งกลางของตารางหน้าต่างคงค่าเดิม ;

นอกเหนือจากนั้น เลือก $L_h = 1$ แล้วไปตรวจสอบทางแนวตั้ง (4.7)

II. การตรวจสอบทางแนวตั้ง :

ถ้า ($Q_2 > T_2$ และ $Q_{-2} > T_2$) หรือ ($Q_1 > T_2$ และ $Q_{-2} > T_2$) หรือ ($Q_2 > T_2$ และ $Q_{-1} > T_2$)

เลือก $L_v = 5$ แล้วทำการเรียงลำดับข้อมูล

ถ้า ($Q_1 > T_1$ และ $Q_{-1} > T_1$) หรือ ($1/2 < Q_1/Q_{-1} < 2$)

เลือก $L_v = 3$ แล้วทำการเรียงลำดับข้อมูล

ถ้า ($T_{-1} < Q_1 < T_1$ และ $-T_1 < Q_{-1} < T_1$ และ $-T_2 < Q_2 < T_2$ และ $-T_2 < Q_{-2} < T_2$)

เลือก $L_v = L_h = 1$ คือ ที่จุดกึ่งกลางของตารางหน้าต่างคงค่าเดิม;

นอกเหนือจากนั้น เลือก $L_v = 1$ แล้วทำการเรียงลำดับข้อมูล (4.8)

ค่า T_1 และ T_2 เป็นค่า threshold ที่ใช้ในการทดสอบว่า ค่า P และ Q เท่าไหร่จึงจะถือว่าเป็นสัญญาณรบกวน โดยจากการทดลองพบว่าค่า T_1และ T_2 ที่ใช้มีค่าอยู่ระหว่าง 16 ถึง 48 และเป็นค่าเฉพาะภาพๆ ไป สำหรับในกรณีของ pepper noise ก็ทำในลักษณะเดียวกันกับ salt noise ยกเว้นเพียงเครื่องหมายของ P และ Q ให้เป็นตรงกันข้ามแล้วทำตามขั้นตอนเดิมทุกประการ ซึ่งจากการทดลองนี้ได้ใช้การจำลองสัญญาณรบกวนแบบผสมทั้ง 2 แบบ ดังนั้นเราจึงทำการกำจัดโดยใช้ขั้นตอนของการกรองมัธยฐานแบบที่ใช้กับ salt noise ก่อนแล้วจึงใช้แบบที่ใช้กับ pepper noise อีกครั้ง

ตัวอย่าง สมมติค่าระดับสีเทาในตารางหน้าต่างเป็นดังภาพที่ 4.5

		55		
		57		
56	57	56	120	57
		56		
		57		

ภาพที่ 4.5 ค่าระดับสีเท่าในหน้าต่าง

โดยใช้ขั้นตอนของการกรอง salt noise และตำแหน่งที่พิจารณาสมมติให้ n = 3 จากสมการที่ 4.6 จะได้ว่า

$x(1) = 56, x(2) = 57, x(3) = 56, x(4) = 120, x(5) = 57$

$y(1) = 55, y(2) = 57, y(3) = 56, y(4) = 56, y(5) = 57$

$p_1 = x(3)-x(4) = 56-120 = -64$

$p_2 = x(4)-x(5) = 120-57 = 63$

$p_{-1} = x(3)-x(2) = 56-57 = -1$

$p_{-2} = x(2)-x(1) = 57-56 = 1$

ในทำนองเดียวกัน

$Q_1 = 0, Q_2 = -1, Q_{-1} = -1, Q_{-2} = 2$

สมมติกำหนดให้ว่า $T_1=30, T_2=40$

นำไปตรวจสอบทางแนวนอน

จากเงื่อนไขแรก (คือ ถ้าเป็นจริงให้เลือก $L_h=5$) ปรากฏว่าไม่เป็นจริง

จากเงื่อนไขต่อมา ปรากฏว่าไม่เป็นไปตามเงื่อนไขต้องตรวจสอบต่อไป

จากเงื่อนไขที่ 3 ก็ไม่เป็นจริงดังนั้น เลือก $L_h=1$

นำไปตรวจสอบทางแนวตั้ง

จากเงื่อนไขแรก ปรากฏว่าไม่เป็นจริง

จากเงื่อนไขที่ 2 ปรากฏว่าก็ไม่เป็นไปตามเงื่อนไขต้อง ตรวจสอบต่อไป

จากเงื่อนไขต่อมา ก็ไม่เป็นจริงดังนั้น เลือก $L_v = 1$

ดังนั้นเราได้ความยาวของตารางหน้าต่างในแนวนอนและแนวตั้งเท่ากัน คือ 1 ซึ่งหมายความว่า ไม่ต้องมีการเรียงลำดับของข้อมูลในตารางหน้าต่าง หรือไม่มีการเปลี่ยนแปลงของค่าที่จุดกำลังพิจารณา (จุดกึ่งกลางหน้าต่าง)

จากตัวอย่างข้างบนจะเห็นว่าที่จุด x(4) คือสัญญาณรบกวนเพราะมีค่ามากกว่าจุดใกล้เคียงมาก ทำให้สามารถรู้ว่าเป็นสัญญาณรบกวนแบบ salt noise ส่วนกรณี pepper noise นั้น จะมีค่าที่ต่ำกว่าค่าใกล้เคียงมากๆ เมื่อดูจากภาพก็จะมองเห็นเป็นจุดที่เด่นชัดมาก เราก็เลือกใช้ขั้นตอนของการกรองแบบที่ใช้กับ salt noise

ถ้าเราใช้การกรองมัธยฐานแบบธรรมดา (ทำการเรียงลำดับข้อมูลเลย $\{55, 56, 56, 56, (57), 57, 57, 57, 120\}$) จะได้ค่าที่จุดพิจารณาเท่ากับ 57 ซึ่งจะเห็นว่าต่างไปจากค่าเดิม (56) เป็นสาเหตุให้ภาพเบลอ

สำหรับการที่เลือกใช้ขนาดของความยาวสูงสุดตารางหน้าต่างเท่ากับ 5 เพราะว่าเป็นขนาดที่กำลังเหมาะสมทั้งในด้านประสิทธิภาพในการกำจัดสัญญาณรบกวนและขั้นตอนในการทำงาน ถ้าใช้ความยาวมากกว่านี้ขั้นตอนจะซับซ้อนและเสียเวลามากขึ้น ในขณะที่ถ้าใช้ขนาดน้อยกว่าประสิทธิภาพก็จะด้อยตาม แต่ถ้ากรณีที่สัญญาณรบกวนหนาแน่นมากๆ ก็สามารถเพิ่มขนาดได้ โดยเพิ่มขั้นตอนขึ้นไปอีก

4.2.2 การกรองมัธยฐานแบบปรับขนาดได้อย่างอัตโนมัติที่ใช้กับ Pepper noise:

สำหรับกรณี pepper noise ซึ่งจะเป็นการกำจัดสัญญาณรบกวนอิมพัลส์แบบจุดดำ ตัวอย่างตามภาพที่ 4.6 ซึ่งจะมีลักษณะตรงกันข้ามกับหัวข้อที่ 4.2.1 เราจะต้องทำการเปลี่ยนเครื่องหมายของ P และ Q ในสมการที่ 4.6 เป็นตรงกันข้ามก่อน ดังนี้

$$
\begin{aligned}
P_j &= -[\, x(n+j-1) - x(n+j) \,] && \text{เมื่อ } j = 1, 2 ; \\
P_j &= -[\, x(n+j+1) - x(n+j) \,] && \text{เมื่อ } j = -1, -2 ; \\
Q_j &= -[\, y(n+j-1) - y(n+j) \,] && \text{เมื่อ } j = 1, 2 ; \\
Q_j &= -[\, y(n+j+1) - y(n+j) \,] && \text{เมื่อ } j = -1, -2 ;
\end{aligned}
\qquad (4.9)
$$

จากตัวอย่างที่แล้ว เราจะได้ค่า P และ Q ดังต่อไปนี้

$P_1 = 64, P_2 = -63, P_{-1} = -1, P_{-2} = 1$

$Q_1 = 0, Q_2 = 1, Q_{-1} = 1, Q_{-2} = -2$

แล้วให้ดำเนินการตรวจสอบค่า ตามสมการที่ 4.7 (ตรวจสอบทางแนวนอน) และสมการที่ 4.8 (ตรวจสอบทางตั้ง) หลังจากนั้นก็ไปหาค่ามัธยฐาน ทำตามขั้นตอนเดิมทุกอย่าง

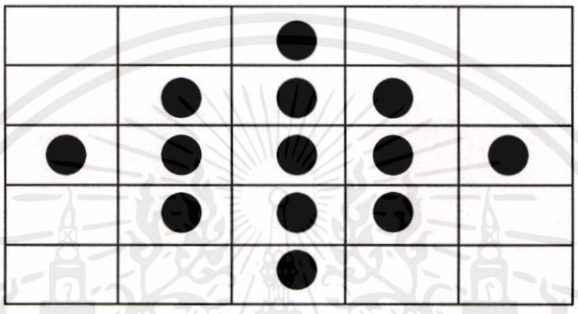
(ก) (ข)

(ค) (ง)

ภาพที่ 4.6 ภาพผลลัพธ์จากภาพ Lena ขนาด 256x256 ที่ถูกรบกวนโดยสัญญาณรบกวนแบบ salt and pepper noise (impulse) ประมาณ 5% (ก)
การกรองมัธยฐานแบบหน้าต่างคงที่ขนาด3x3 (ข)
การกรองมัธยฐานแบบหน้าต่างคงที่ขนาด5x5 (ค)
การกรองมัธยฐานที่ปรับขนาดหน้าต่างได้อย่างอัตโนมัติ เมื่อใช้ T_1=16 และ T_2=45 (ง)

4.3 การขยายหน้าต่างการกรองมัธยฐานแบบปรับขนาดได้อย่างอัตโนมัติ

ถ้าสัญญาณรบกวนแบบอิมพัลส์มีเปอร์เซ็นต์สูงมากและต้องการคุณภาพของภาพให้ดีขึ้น เราสามารถที่จะปรับขยายขนาดหน้าต่างให้ใหญ่ขึ้นโดยเพิ่มเงื่อนไขการเช็คในแนวเฉียงทั้ง 2 ด้านของ filter window ดูได้จากภาพที่ 4.7 จะกลายเป็นว่าเราสนใจด้านทั้ง 4 ของ filter window (Four major correlation direction) [11] ขนาดเงื่อนไขที่เพิ่มเข้ามานี้จะกระทำก่อนการตัดสินใจในแนวนอนและแนวตั้ง ขนาดหน้าต่างสูงสุดดูจากพิกเซลในแนว 45 องศา ถูกกำหนดโดย



ภาพที่ 4.7 การขยายหน้าต่างการกรองมัธยฐานแบบปรับขนาดได้

$$\mu(n-2), \mu(n-1), \mu(n), \mu(n+1), \mu(n+2);$$

พิกเซลในแนว 135 องศา ถูกกำหนดโดย

$$Z(n-2), Z(n-1), Z(n), Z(n+1), Z(n+2);$$

เมื่อ $\mu(n) = Z(n)$ เมื่อพิกเซลกึ่งกลาง (central pixel) เรากำหนดให้

$$\begin{aligned} r_j &= \mu(n+j-1) - \mu(n+j), && \text{เมื่อ } j = 1,2; \\ r_j &= \mu(n+j+1) - \mu(n+j), && \text{เมื่อ } j = -1,-2; \\ S_j &= Z(n+j-1) - Z(n+j), && \text{เมื่อ } j = 1,2; \\ S_j &= Z(n+j+1) - Z(n+j), && \text{เมื่อ } j = -1,-2; \end{aligned} \qquad (4.10)$$

ให้ L_μ, L_Z แทนความยาวของขนาดหน้าต่างในแนวทแยง 45 องศาและ 135 องศาตามลำดับ ขบวนการตรวจเช็คมีดังต่อไปนี้

ในแนวการตรวจ 45 องศา

ถ้า ($r_1 > T_1$ และ $P_{-1} > T_1$) หรือ ($r_2 > -T_2$ และ $r_{-2} > -T_2$),

เลือก $L_\mu = 3$ แล้วเช็คในแนว 135 องศาต่อ

ถ้า $(-T_1 < r_1 < T_1$ และ $-T_1 < r_{-1} < T_1)$ และ $-T_2 < r_2 < T_2$ และ $-T_2 < r_{-2} < T_2$

เลือก $L_\mu = L_Z = L_h = L_v = 1$, หรือพิกเซลกึ่งกลาง (ค่าเดิม) แล้วไม่ต้องเช็คด้านอื่น

กรณีอื่นให้เลือก $L_\mu = 1$ และไปเช็คในแนว 135 องศาต่อ

ในแนวการตรวจเช็ค 135 องศา

ถ้า $(S_1 > T_1$ และ $S_{-1} > T_1)$ หรือ $(S_2 > -T_2$ และ $S_{-2} > -T_2)$,

เลือก $L_\mu = 3$ แล้วเช็คในแนวนอนต่อ

ถ้า $(-T_1 < S_1 < T_1$ และ $-T_1 < S_{-1} < T_1)$ และ $-T_2 < S_2 < T_2$ และ $-T_2 < S_{-2} < T_2$

เลือก $L_\mu = L_Z = L_h = L_v = 1$, หรือพิกเซลกึ่งกลางโดยไม่ต้องตรวจสอบด้านอื่น

กรณีอื่นให้เลือก $L_\mu = 1$ และไปเช็คในแนวนอนต่อ (4.11)

อัลกอริทึมในการเช็คแนวนอนและแนวตั้งยังคงเหมือนเดิมที่ได้กล่าวไว้แล้วในตอนต้น

บทที่ 5
การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง

จากพื้นฐานการกรองมัธยฐานในบทที่ 3 และการดัดแปลง ปรับปรุงเทคนิควิธีการกรองมัธยฐานในบทที่ 4 ซึ่งทำให้คุณภาพของภาพดีขึ้น ตามลำดับ ในบทนี้จะนำเสนอวิธีการกรองมัธยฐานมัลติเชลล์แบบดัดแปลง (Adaptive multi-shell median filter) ที่ให้ภาพผลลัพธ์มีคุณภาพดีและมีความยุ่งยากในอัลกอริธึมไม่มาก

5.1 พื้นฐานการกรองมัธยฐานแบบมัลติ-เชลล์

การกรองมัธยฐานแบบมัลติเชลล์ถูกนำเสนอโดย J.S. Jimmy Li [12] ซึ่งมีคุณสมบัติที่ดี คือ ให้ภาพผลลัพธ์ที่มีความคมชัดสูง สามารถรักษารายละเอียดเล็ก ๆ น้อย ๆ ได้ดี เช่น ภาพที่ประกอบด้วยเส้นที่บาง ๆ หรือขอบมุมที่มีความคม ความหมายของการกรองมัธยฐานแบบมัลติเชลล์ คือ การออกแบบลักษณะของหน้าต่างในการกรอง (Filter window) ให้มีหลายรูปแบบซ้อนทับกันอยู่จากนั้นให้หาค่าสูงสุดและต่ำสุดของแต่ละหน้าต่าง แล้วนำค่าสูงสุดและต่ำสุดเหล่านั้นมาทำการหาค่ามัธยฐานร่วมกับค่าที่จุดกึ่งกลาง (Central pixel) อีกครั้งหนึ่ง

ก่อนอื่นลองพิจารณาค่าของตัวเลข 3 ค่าในแบบ 1 มิติต่อไปนี้ สมมติให้ a_1, a_2, a_3 เป็นค่าระดับสีเทา(Gray level) ตามสมการมัธยฐาน คือ

$$median\{a_1, a_2, a_3\} = median\{\min[a_1, a_3], a_2, \max[a_1, a_3]\} \qquad (5.1)$$

โดยที่ a_2 เป็นตัวอย่างกลางที่ถูกพิจารณา (central sample) ส่วน $[a_1, a_3]$ เป็นเซตของตัวอย่างที่อยู่รอบ ๆ ตัวอย่างกลาง หรือ เป็นเปลือกหรือเชลล์ (Shell) ของการหาค่ามัธยฐาน ในขั้นตอนนี้เราจะต้องหาค่าน้อยที่สุดในเชลล์ ($\min[a_1, a_3]$) และมากที่สุดในเชลล์ ($\max[a_1, a_3]$) จากนั้นจึงไปทำการค่าหามัธยฐานตามสมการ 5.1

จากสมการที่ 5.1 ซึ่งเป็นแบบ 1 มิติ เราสามารถขยายผลโดยการหมุนสมการ 5.1 รอบตัวเองเพื่อทำเป็นแบบ 2 มิติ ขนาดหน้าต่างจตุรัส 3x3 โดยกำหนดให้ $y_{m,n}$ คือ เอาท์พุทของการกรองที่ตำแหน่ง m,n และมี $a_{m,n}$ เป็นตัวอย่างกลาง และ S_1 คือ เซตของสมาชิกทั้งหมดที่อยู่รอบตัวอย่างกลาง

$$S_1 = \{a_{m-1,n-1} \ , a_{m-1,n} \ , a_{m-1,n+1} \ , \\ a_{m,n-1} \ , \quad , a_{m,n+1} \ , \\ a_{m+1,n-1}, a_{m+1,n} \ , a_{m+1,n+1}\} \qquad (5.2)$$

S_1 คือเซลล์ที่ 1 ของเซตสมาชิกที่มีอยู่รอบตัวอย่างกลาง สมการเอาท์พุทสามารถเขียนได้ดังนี้

$$Y_{m,n} = median(\min(S_1), a_{m,n}, \max(S_1)) \qquad (5.3)$$

ผลของสมการที่ 5.3 กับภาพ LENA ที่มีสัญญาณรบกวนแบบอิมพัลซ์ 5%

จากสมการที่ 5.3 ชี้ให้เห็นว่าถ้าตัวอย่างกลาง $(a_{m,n})$ เป็นอิมพัลซ์เดี่ยว ๆ คือ มีค่าแตกต่างจากสมาชิกที่อยู่ในเซลล์มาก ๆ (อาจจะมากว่ามาก ๆ หรือน้อยกว่ามาก ๆ) เมื่อผ่านการหาค่ามัธยฐานจะถูกแทนที่ด้วยสมาชิกที่อยู่ในเซลล์ตามสมการที่ 5.3 แต่ในอีกทางหนึ่งถ้าอิมพัลซ์ติดกันเป็นลักษณะ 2 พิกเซล สมการที่ 5.3 ก็จะไม่สามารถกำจัดออกไปได้ยังคงเหลืออิมพัลซ์นี้ติดมาด้วย เราสามารถอธิบายวิธีการทำงานของโปรแกรมนี้ได้ตามตัวอย่างที่ 5.1 และ 5.2

ตัวอย่างที่ 5.1 เราสามารถอธิบายเพื่อประกอบความเข้าใจดังต่อไปนี้ สมมติให้ค่าระดับเทาในฟิลเตอร์วินโดว์ขนาด 3x3 จตุรัส ประกอบด้วยค่าต่าง ๆ ดังตารางข้างล่าง สมมุติให้ $a_{m,n}$ เป็นอิมพัสซ์ซึ่งมีค่าแตกต่างจากค่าสมาชิกในเซลล์มาก ๆ

70	74	80
70	250	61
70	73	65

S_1	S_1	S_1
S_1	$a_{m,n}$	S_1
S_1	S_1	S_1

เราจะได้ว่า $\min(S_1) = 61$

$\max(S_1) = 80$

$a_{m,n} = 250$

$Y_{m,n} = median(61,250,80)$

$= 80$

ซึ่งแสดงให้เห็นว่าอิมพัลซ์นอยส์ได้ถูกกำจัดออกไปแล้ว

(ก)

(ข)

(ค)

(ง)

ภาพที่5.1 ภาพผลลัพธ์ของการกรอง ภาพ LENA ต้นฉบับ (ก)
ภาพ LENA ที่มีสัญญาณรบกวนอิมพัลซ์ 5% (ข)
ภาพ LENA ที่ผ่านการกรองมัธยฐานด้วยขนาดหน้าต่าง 3x3 (ค)
ภาพ LENA ที่ผ่านการกรองมัธยฐานแบบมัลติเชลล์ (ง)

ตัวอย่างที่ 5.2_ในกรณีอิมพัลซ์ 2 จุดติดกัน ถ้า $a_{m,n}$ เป็นอิมพัลซ์และ $a_{m-1,n-1}$

251	74	80
70	250	61
73	73	65

เราจะได้ว่า $\min(S_1) = 61$

$\max(S_1) = 251$

$a_{m,n} = 250$

$Y_{m,n} = median(61,251,250)$

$= 250$

ซึ่งแสดงให้เห็นว่าเอาท์พุทของฟิลเตอร์วินโดว์นี้ยังคงรักษาความเป็นอิมพัลซ์อยู่ หรือกล่าวอีกนัยหนึ่ง คือ ไม่สามารถกำจัดอิมพัลซ์ออกไปได้

ดังนั้นในการที่จะกำจัดอิมพัลซ์ที่ติดมานั้นขนาดของหน้าต่างก็จะต้องใหญ่ขึ้นดังเช่นที่กล่าวไว้บ้างแล้วในบทที่ 4 เราสามารถขยายหน้าต่างโดยเพิ่มสมาชิกได้ด้วยสมการต่อไปนี้

$$median\{a_1,a_2,a_3,a_4,a_5\} = median\{\min(a_1,a_5),\min(a_2,a_4),a_3,\max(a_2,a_4),\max(a_1,a_5)\} \quad (5.4)$$

เมื่อ a_3 คือตัวอย่างกลางที่ถูกพิจารณาและ (a_2,a_4) คือเซลล์ที่ 1 (a_1,a_5) คือเซลล์ที่ 2 ที่อยู่ล้อมรอบตัวอย่างกลาง เราสามารถขยายผลตัวนี้ให้เป็นแบบ 2 มิติ โดยมีขนาดหน้าต่างจตุรัส 5x5 ซึ่งจะสามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์ได้มากขึ้น ขนาดหน้าต่างหรือเซลล์นี้สามารถขยายตัวออกไปได้อีกในกรณีที่เราต้องการ เช่น 7x7 หรือมากกว่านั้น เซลล์ที่เกิดขึ้นเราจะเรียงตามลำดับว่า 1,2, _ _ _ เราแทนชื่อเซลล์ต่าง ๆ ด้วย S_j เมื่อ j เป็นเลขจำนวนเต็ม

$$S_j = \{ \quad a_{m-j,n-j} \;,\; a_{m-j,n-j+1} \;,\; \text{---------},\; a_{m-j,n+j} \;,$$
$$a_{m-j+1,n-j} \;,\; a_{m-j+1,n-j+1} \;,$$
$$\text{----------} \;,\; \text{----------} \;,$$
$$a_{m+j-1,n-j} \;,\; a_{m+j-1,n-j+1} \;,$$
$$a_{m+j,n-j} \;,\; a_{m+j,n-j+1} \;,\; \text{---------},\; a_{m+j,n+j} \; \} \quad (5.5)$$

เมื่อ $1 \le j \le N$ เราสามารถคำนวณจำนวนสมาชิกในแต่ละเชลล์ได้ด้วยสมการ 5.6

$$\left|S_j\right| = (2_{j+1})^2 - (2_{(j-1)+1})^2 = 8_j \tag{5.6}$$

จำนวนสมาชิกทั้งหมดในหน้าต่างขนาด N คือ $(2N+1)^2$

เอาท์พุทของการกรองมัลติเชลล์ คือ

$$Y_{m,n} = median(W) \tag{5.7}$$

เมื่อ

$$W = (\min(S_N), \_\_\_, \min(S_1), a_{m,n}, \max(S_1), \_\_\_, \max(S_n)) \tag{5.8}$$

$|W|$ จะมีค่าเท่ากับ $(2N+1)$ ซึ่งเป็นความยาวหนึ่งด้านของฟิลเตอร์วินโดว์ในแบบ 2 มิติ ถ้าจำนวนอิมพัลส์ไม่มากกว่า N ตัว การกรองมัธยฐานมัลติเชลล์นี้จะสามารถกำจัดสัญญาณรบกวนออกไปได้หมด นอกจากนี้ค่า N ยังแสดงถึงจำนวนของเชลล์ที่เกิดขึ้น เช่น N=2 จำนวนเชลล์ที่เกิดขึ้นจะมี 2 เชลล์ ตามตารางที่ 5.1

ตารางที่ 5.1 จำนวนเชลล์ที่เกิดขึ้นจากการกรองมัธยฐาน

S_2	S_2	S_2	S_2	S_2
S_2	S_1	S_1	S_1	S_2
S_2	S_1	a_{mn}	S_1	S_2
S_2	S_1	S_1	S_1	S_2
S_2	S_2	S_2	S_2	S_2

S_1 แสดงจำนวนสมาชิกทั้งหมดในเชลล์ที่1

S_2 แสดงจำนวนสมาชิกทั้งหมดในเชลล์ที่2

a_{mn} คือ จุดพิกเซลที่จุดกึ่งกลาง

(ก)

(ข)

ภาพที่ 5.2 ภาพผลลัพธ์ของการกรอง ภาพ LENA ที่มีสัญญาณรบกวนอิมพัลซ์ 5% (ก) ภาพ LENA ที่ผ่านการกรองมัธยฐานมัลติเชลล์ แบบมี 2 เชลล์ (ข)

5.2 การกรองมัธยฐานมัลติเชลล์สำหรับ Missing Line Noise

Missing Line Noise เกิดจากการสูญเสียสัญญาณภาพเป็นแนวเส้นจากการส่งสัญญาณหรืออาจเกิดจากความผิดพลาดในการถอดรหัสและไม่สามารถทำให้กลับคืนให้ดีดังเดิมได้ การสูญเสียสัญญาณภาพเป็นแนวเส้นนี้จะเกิดเฉพาะในแนวนอนเท่านั้น เนื่องจากการส่งสัญญาณภาพโดยทั่ว ๆ ไปจะส่งในแนวนอนก่อน แล้วจึงนำสัญญาณในแนวนอนมาประกอบกันจนเกิดเป็นภาพที่สมบูรณ์ เราสามารถออกแบบฟิลเตอร์วินโดว์เพื่อกำจัดสัญญาณรบกวนแบบ Missing Line ได้ตามสมการที่ 5.9 ดังนี้

$$S_j = \left\{ \begin{array}{l} a_{m-j,n-j}\ ,\ a_{m-j,n-j+1},\ \cdots\ ,\ a_{m-j,n+j}, \\ a_{m-j+1,n-j},\ a_{m-j+1,n+j}, \\ \quad \cdots\ ,\ \quad \cdots\ , \\ a_{m-1,n-1}\ ,\ a_{m-1,n+j}\ , \\ a_{m+1,n-j}\ ,\ a_{m+1,n+j}\ , \\ \quad \cdots\ ,\ \quad \cdots\ , \\ a_{m+j-1,n-j},\ a_{m+j-1,n+j}, \\ a_{m+j,n-j},\ a_{m+j,n-j+1},\ \cdots,\ a_{m+j,n+j} \end{array} \right\} \tag{5.9}$$

และ $1 \le j \le N; |S_j| = 8_j - 2$

สมมติว่า N=1 ขนาดฟิลเตอร์วินโดว์ คือ 3x3, N=2 ขนาดฟิลเตอร์วินโดว์ คือ 5x5 เราสามารถแสดงจำนวนสมาชิกได้ดังตารางข้างล่างนี้ พิกเซลในแนวนอนที่ตรงกับตัวอย่างกลางจะไม่ถูกรวมเข้าในฟิลเตอร์วินโดว์ ซึ่งจะทำให้สามารถกำจัดสัญญาณรบกวนแบบ missing line noise ไปได้ (ตารางที่ 5.2)

ตารางที่ 5.2 สมาชิกในหน้าต่างการกรองมัธยฐานมัลติเซลล์สำหรับ Missing Line Noise แบบ 2 เซลล์

S_2	S_2	S_2	S_2	S_2
S_2	S_1	S_1	S_1	S_2
		a_{mn}		
S_2	S_1	S_1	S_1	S_2
S_2	S_2	S_2	S_2	S_2

S_1 แสดงจำนวนสมาชิกทั้งหมดในเซลล์ที่1

S_2 แสดงจำนวนสมาชิกทั้งหมดในเซลล์ที่2

a_{mn} คือ จุดพิกเซลที่จุดกึ่งกลาง

ตารางที่ 5.3 สมาชิกในหน้าต่างการกรองมัธยฐานมัลติเซลล์สำหรับ Missing Line Noise แบบ 1 เซลล์

S_1	S_1	S_1
	a_{mn}	
S_1	S_1	S_1

S_1 แสดงจำนวนสมาชิกทั้งหมดในเซลล์ที่1

a_{mn} คือ จุดพิกเซลที่จุดกึ่งกลาง

(ก)

(ข)

(ค)

(ง)

ภาพที่ 5.3ภาพผลลัพธ์ของการกรองที่มีสัญญาณรบกวน (Missing line noise) อิมพัลซ์ 5% (ก)
ภาพ LENA ที่ผ่านการกรองมัธยฐานด้วยขนาดหน้าต่าง 3x3 (ข)
ภาพ LENA ที่ผ่านการกรองมัธยฐานมัลติเชลล์ แบบ 1 เชลล์ (ค)
ภาพ LENA ที่ผ่านการกรองมัธยฐานมัลติเชลล์ แบบ 2 เชลล์ (ง)

5.3 การกรองมัธยฐานมัลติเชลล์แบบดัดแปลงแบบที่ 1 (Adaptive Multi-Shell Median Filter First Algorithms)

หัวข้อที่แล้วได้แนะนำถึงวิธีการกรองมัธยฐานแบบมัลติเชลล์ หัวข้อนี้จะขอเสนอการกรองมัธยฐานแบบดัดแปลง อ้างถึง Ioannis Pitas [14] ที่ได้เสนอหลายๆ อัลกอริธึม ในการคำนวณเพื่อจัดเรียงค่าสูงสุดต่ำสุด (Fast algorithms for running ordering and max/min calculation) จากพื้นฐานดังกล่าวได้ถูกนำมาประยุกต์และดัดแปลงเพื่อเพิ่มประสิทธิภาพ และความสามารถในการกำจัดสัญญาณรบกวนแบบอิมพัลส์ ทำให้ภาพผลลัพธ์มีความผิดเพี้ยนจากภาพต้นแบบน้อยกว่าวิธีการกรองมัธยฐานมัลติเชลล์แบบธรรมดา หรือกล่าวอีกนัยหนึ่งคือ การกรองมัธยฐานมัลติเชลล์แบบดัดแปลงให้ผลลัพธ์ดีกว่านั่นเอง

จากพื้นฐานการกรองมัธยฐานมัลติเชลล์ในหัวข้อนี้จะเลือกรูปแบบหน้าต่างที่ใช้เพื่อกำจัดสัญญาณรบกวนแบบขาดหายไปบางเส้นมาประยุกต์ดังนี้

$a_{m-1,n-1}$	$a_{m-1,n}$	$a_{m-1,n+1}$
	$a_{m,n}$	
$a_{m+1,n-1}$	$a_{m+1,n}$	$a_{m+1,n+1}$

S	S	S
S	S	S

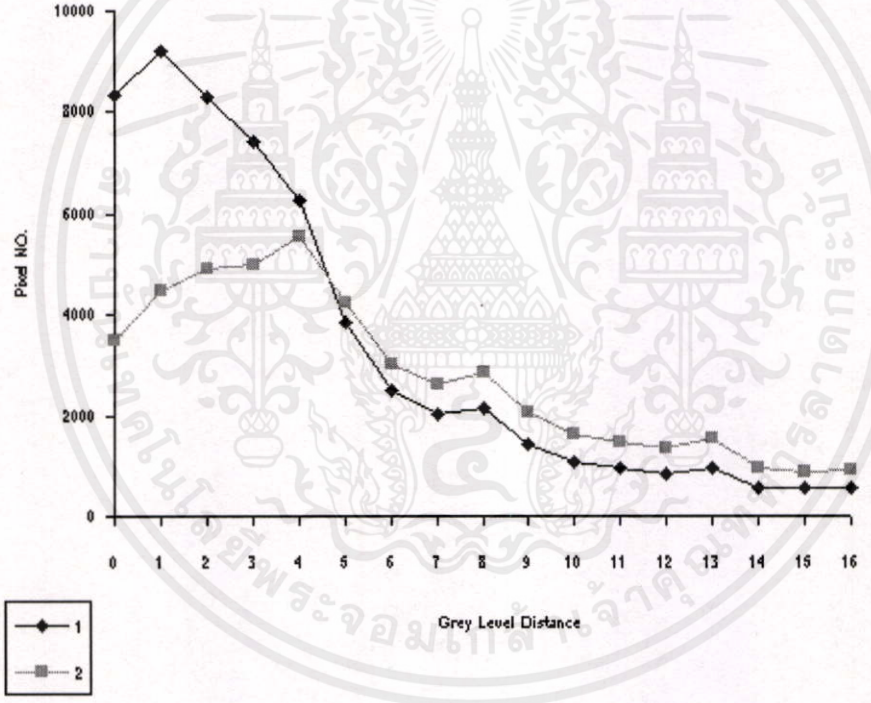
สมการเอาท์พุทที่ตำแหน่ง m,n ของการกรองมัธยฐานมัลติเชลล์ คือสมการที่ 5.3 เรานำมาเขียนใหม่เพื่อให้ง่ายต่อการเข้าใจ ดังต่อไปนี้

$$Y_{m,n} = \begin{cases} Max(S) & if\ a_{m,n} > Max(S) \\ a_{m,n} & if\ Min(S) < a_{m,n} < Max(S) \\ Min & if\ a_{m,n} < Min(S) \end{cases} \tag{5.10}$$

จากการค้นคว้าวิจัยเพื่อที่จะพัฒนาให้การกรองมัธยฐานมีประสิทธิภาพที่ดียิ่งขึ้น พบว่าค่าระดับเทาในตำแหน่ง $a_{m-1,n}$, $a_{m,n}$, $a_{m+1,n}$ จะมีค่าใกล้เคียงกันมากกว่าในตำแหน่งอื่น ๆ เช่น $a_{m-1,n-1}$, $a_{m-1,n+1}$, $a_{m+1,n-1}$, $a_{m+1,n+1}$ ดังนั้นเราจะกำหนดสมการเอาท์พุท ของการกรองมัธยฐานมัลติเชลล์แบบดัดแปลง ใหม่ คือ

$$Y_{m,n} = \begin{cases} Max(a_{m-1,n}, a_{m+1,n}) & if\ a_{m,n} > Max(S) \\ a_{m,n} & if\ Min(S) < a_{m,n} < Max(S) \\ Min(a_{m-1,n}, a_{m+1,n}) & if\ a_{m,n} < Min(S) \end{cases} \tag{5.11}$$

จากเหตุผลที่ว่าค่าระดับเทาของ $a_{m-1,n}$, $a_{m,n}$, $a_{m+1,n}$ มีค่าใกล้เคียงกันทำให้วิธีการนี้มีผลลัพท์ของการกรองที่ดีขึ้นกว่าเดิม เพื่อประกอบการพิจารณาเราได้ทำการทดลองเพื่อหาค่าความแตกต่างระดับเท่ากับภาพ Lena, Girl,...ฯลฯ มากกว่า 10 ภาพ ค่าความน่าจะเป็นของระยะห่างค่าระดับเทา (Probability of the gray level distance) ที่ทำการเปรียบเทียบระหว่างค่าที่จุดกึ่งกลาง ($a_{m,n}$) กับค่าที่อยู่ในแนวเส้นตรง($a_{m-1,n}$, $a_{m+1,n}$) จะมีความแตกต่างของ Gray Level Distance ที่ต่ำกว่า 10 สูงกว่าการเปรียบเทียบระหว่างค่าที่จุดกึ่งกลางกับค่าโดยรอบ (S) อยู่ 8% โดยประมาณ (จากภาพที่ 5.4) จากเหตุผลนี้ถ้า $a_{m,n}$ ถูกรบกวนโดยอิมพัลซ์นอยส์จึงน่าจะดีกว่าที่ $a_{m,n}$ จะถูกแทนที่ด้วย.. $a_{m-1,n}$ หรือ $a_{m+1,n}$ มากกว่าค่าของMax(S) หรือ Min (S)



ภาพที่ 5.4 ความสัมพันธ์ระหว่างการกระจายของสีเทากับจำนวนพิกเซล

จากภาพที่ 5.4 เมื่อกำหนดให้ 1 คือ $\left|a_{m,n} - Max(a_{m-1,n}\,|\,a_{m+1,n})\right|$

และ 2 คือ $\left|a_{m,n} - Max(S)\right|$

แสดงค่า ระยะห่างค่าระดับเทา (Grey level distance) ของภาพ LENA

เราสามารถอธิบายวิธีการกำจัดสัญญาณรบกวนได้ตามภาพที่ 5.5 ดังต่อไปนี้

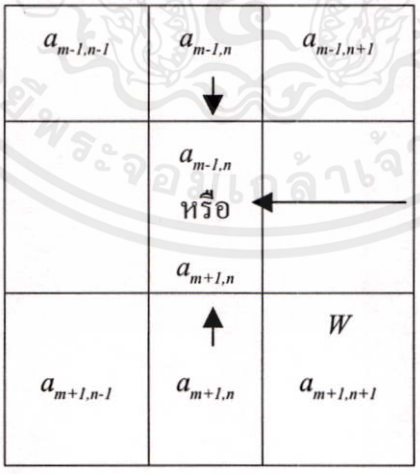
$a_{m-1,n-1}$	$a_{m-1,n}$	$a_{m-1,n+1}$
	W	
$a_{m+1,n-1}$	$a_{m+1,n}$	W $a_{m+1,n+1}$

ภาพที่ 5.5 วิธีการกำจัดสัญญาณรบกวน

ให้ W คือ White impulse Noise (อิมพัลซ์นอยส์แบบจุดขาว)

$a_{m-1,n-1}$,..... $a_{m+1,n+1}$.คือสมาชิกในเซลล์

จากภาพที่ 5.5 ถ้าอิมพัลซ์ปรากฏที่จุดกึ่งกลางหน้าต่างพร้อมกับที่มุมใดมุมหนึ่งหรือมากกว่าของหน้าต่าง (ในภาพยกตัวอย่าง W ที่ปรากฏมุมล่างขวา) การกรองมัธยฐานมัลติเซลล์แบบที่กล่าวไว้ในตอนแรกจะไม่สามารถกำจัดอิมพัลซ์นอยส์ออกไปได้ ดังที่อธิบายไว้ในหัวข้อที่แล้ว ส่วนวิธีการกรองมัธยฐานมัลติเซลล์แบบดัดแปลงจะสามารถแทนที่อิมพัลซ์นอยส์ได้ด้วยค่า $a_{m-1,n}$ หรือ. $a_{m+1,n}$.ซึ่งผลลัพธ์ที่ออกมาก็ย่อมดีกว่า (ภาพที่ 5.6)



ภาพที่ 5.6 วิธีการกรองมัธยฐานมัลติเซลล์แบบดัดแปลง

จากการคำนวณทางสถิติและการหาค่า Gray Level Distance ของภาพหลาย ๆ ภาพเราจะได้ว่า

$$Max(a_{m-1,n}, a_{m+1,n}) \le Max(S) \tag{5.12}$$

$$Min(a_{m-1,n}, a_{m+1,n}) \ge Min(S) \tag{5.13}$$

จากสมการ 5.12 และ 5.13 เราจะได้สมการใหม่ดังนี้

$$\left|a_{m,n} - Min(S)\right| \le \left|a_{m,n} - (a_{m-1,n} | a_{m+1,n})\right| \le \left|a_{m,n} - Max(S)\right|$$

ถ้า $Min(S) \le a_{m,n} \le Max(S)$ (5.14)

ในทางปฏิบัติกับข้อมูลภาพจริงสมการที่ 5.14 จะมีส่วนที่ไม่ถูกต้องเพียง 2.5% จากการทดสอบผลลัพธ์ ซึ่งจากมุมมองทางสถิติแล้วสมการ 5.14 จะถือได้ว่ามีความถูกต้อง สมการนี้จะบอกได้ว่าการแทนที่ $a_{m,n}$ ด้วย.. $a_{m-1,n}$ หรือ $a_{m+1,n}$ จะให้ผลลัพธ์ที่ดีกว่าค่าของMax(S) หรือ Min (S)

จากกราฟของ Gray Level Distance เราจะพบว่าค่าความแตกต่างของระดับเท่าในช่วง 0-16 มีจุดภาพอยู่เป็นสูงกว่า 90% เพื่อช่วยให้การกรองมีประสิทธิภาพมากยิ่งขึ้นไปอีกหรือลดการถูกแทนที่โดยไม่จำเป็น (ลดการเบลอของภาพ) เราจะเพิ่มการเช็คเทรชโฮลด์ (Threshold) เข้าไปในสมการ 5.11 เพื่อเพิ่มเงื่อนไขการเช็กลดการผิดพลาดและเป็นการพยายามที่จะรักษารายละเอียดเล็ก ๆ น้อยของภาพ

$$Y_{m,n} = \begin{cases} Max(a_{m-1,n}, a_{m+1,n}) & if\ a_{m,n} - Max(S) > 16 \\ Min(a_{m-1,n}, a_{m+1,n}) & if\ Min(S) - a_{m,n} > 16 \\ a_{m,n} & \text{ในกรณีอื่นๆ} \end{cases} \tag{5.15}$$

การแทนที่ของการกรองจะเกิดขึ้นก็ต่อเมื่อระยะห่างระหว่าง $a_{m,n}$ และMax(S) หรือ Min(S) มากกว่า 16 (ภาพที่ 5.7)

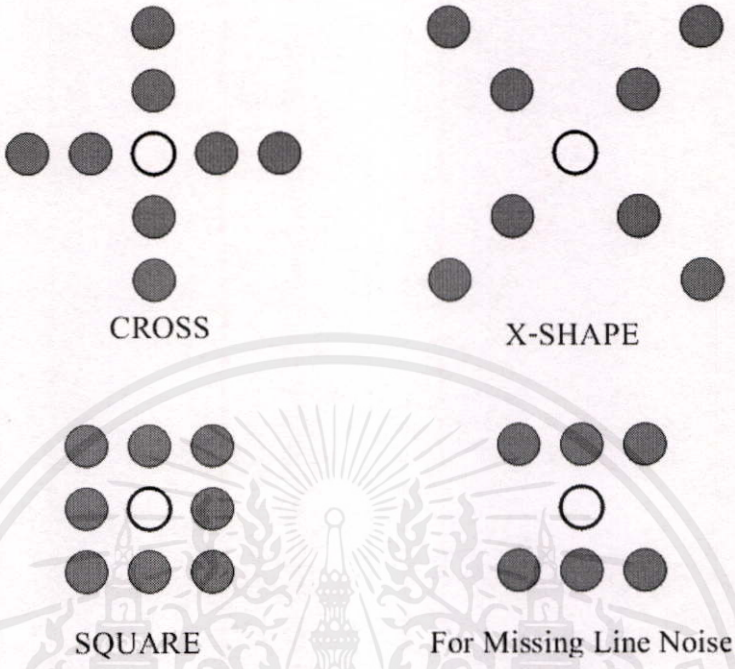
(ก)

(ข)

ภาพที่ 5.7 ภาพผลลัพธ์ จากภาพ LENA ที่มีสัญญาณรบกวนอิมพัลซ์ 5%
การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง ตามสมการ 5.11 (ก)
การกรองมัธยฐานมัลติเชลล์แบบดัดแปลง ตามสมการ 5.15 ที่มีการตรวจสอบค่าเทรช โฮลด์ที่16 (ข)

5.4 การกรองมัธยฐานมัลติเชลล์แบบดัดแปลงแบบที่ 2 (Adaptive Multi-Shell Median filter Second Algorithms)

เป็นการดัดแปลงจากการกรองมัธยฐานมัลติเชลล์เพื่อให้มีคุณสมบัติพิเศษบางประการที่ดีขึ้น คือ ให้มีความคมชัดสูง กำจัดสัญญาณรบกวนได้มาก จากข้อดีของการกรองมัธยฐานมัลติเชลล์ที่ให้ความคมชัดสูง แต่ไม่สามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์ที่อยู่ติดกัน 2 จุดได้ วิธีการนี้จะนำเทคนิคการกรองมัธยฐานแบบธรรมดามาดัดแปลงร่วมเพื่อใช้กับการกรองมัธยฐานมัลติเชลล์ การออกแบบเชลล์ เราควรจะเลือกเชลล์ให้สอดคล้องกับลักษณะเด่นของภาพ เพื่อให้ได้ภาพผลลัพธ์ที่ต้องการทำการกรองมีความคมชัดและผิดเพี้ยนน้อยที่สุด โดยตัวอย่างของลักษณะเชลล์แบบต่าง ๆ ที่สามารถนำมาใช้ในการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงแบบที่ 2 นี้ แต่ละเชลล์จะประกอบไปด้วย พิกเซลที่จุดกึ่งกลางหน้าต่าง และสมาชิกที่อยู่ล้อมรอบทั้งหมด (ภาพที่ 5.8)



ภาพที่ 5.8 ลักษณะเซลล์แบบต่าง ๆ ที่สามารถนำมาใช้ในการกรองมัธยฐานมัลติเซลล์แบบดัดแปลงแบบที่ 2

เราสามารถเขียนสมการเอาพุทท์ได้ดังต่อไปนี้

$$Y_{m,n} = \begin{cases} a_{m,n} & if\ Min(S) < a_{m,n} < Max(S) \\ median(S) & \text{ในกรณีอื่นๆ} \end{cases} \qquad (5.16)$$

เมื่อ $a_{m,n}$ คือพิกเซลที่จุดกึ่งกลางหน้าต่าง

S คือเซลล์ หรือเซตของสมาชิกทั้งหมดที่มีอยู่รอบพิกเซลที่จุดกึ่งกลางหน้าต่าง

(ก)

(ข)

(ค)

(ง)

ภาพที่ 5.9 ภาพภาพ LENA ที่มีสัญญาณรบกวนอิมพัลซ์ 5% (ก)
การกรองมัธยฐานแบบหน้าต่างคงที่3x3 (ข)
การกรองมัธยฐานมัลติเซลล์แบบดัดแปลง 2nd algorithm Square shape (ค)
การกรองมัธยฐานมัลติเซลล์แบบดัดแปลง 2nd algorithm Cross shape (ง)

บทที่ 6

ผลการทดลอง

ผลการทดลองในบทนี้จะแสดงให้เห็นถึงการเปรียบเทียบประสิทธิภาพของการกำขัดสัญญาณรบกวนแบบอิมพัลส์ ของการกรองมัธยฐานวิธีต่าง ๆ ที่ได้กล่าวไว้ตั้งแต่บทที่ 3 ถึงบทที่ 5 โดยได้ทดลองกับภาพต้นแบบที่เป็นที่นิยมใช้กัน คือ LENA, CMAN และ GIRL ซึ่งทั้ง 3 ภาพเป็นภาพระดับเทา 256 ระดับขนาด 256x256 พิกเซล การเปรียบเทียบจะใช้ 2 วิธี คือ ดูภาพผลลัพธ์ด้วยสายตาและใช้วิธีเปรียบเทียบทางการคำนวณซึ่งจะใช้ 2 วิธีที่เป็นดัชนีชี้วัด คือ สมการที่ 2.3 ค่าเฉลี่ยผิดพลาดกำลังสอง (Mean square error : MSE) และสมการที่ 2.4 ค่าเฉลี่ยผิดพลาดทางขนาด (Mean absolute error : MAE) การทดลองจะนำภาพต้นแบบมาใส่สัญญาณรบกวนแบบอิมพัลส์ 5% แล้วใช้การมัธยฐานแบบหน้าต่างคงที่ การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ การกรองมัธยฐานมัลติ-เชลล์ การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ การกรองมัธยฐานมัลติ-เชลล์แบบดัดแปลงที่ 1^{st} และการกรองมัธยฐานมัลติ-เชลล์แบบดัดแปลงที่ 2^{nd} ภาพผลการทดลองของภาพ CMAN ได้แสดงในภาพที่ 6.1 และภาพ GIRL ได้แสดงไว้ในภาพที่ 6.2

นำภาพผลลัพธ์ทั้งหมดของภาพ CMAN, GIRL และ LENA (ภาพผลลัพธ์ LENA ได้ถูกแสดงอยู่ส่วนต่าง ๆ ของวิทยานิพนธ์แล้ว) มาทำการหาค่า MSE และ MAE เทียบกับภาพต้นแบบดังแสดงในตารางที่ 6.1, 6.2 และ 6.3

(ก)

(ข)

(ค)

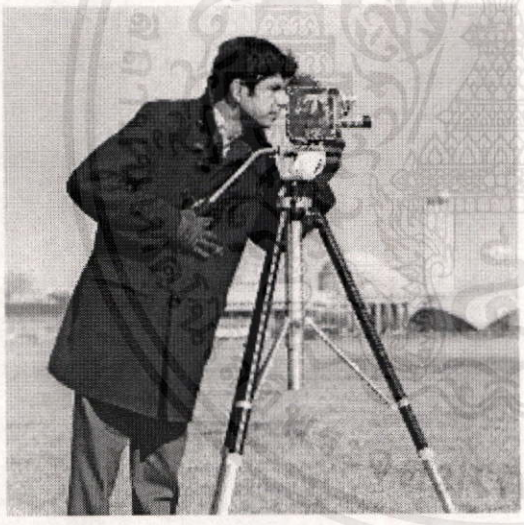
(ง)

ภาพที่ 6.1 ภาพผลลัพธ์ของการกรอง ภาพ CMAN ต้นฉบับ (ก)
ภาพ CMAN ที่มีสัญญาณรบกวนอิมพัลส์ 5% (ข)
ภาพ CMAN ที่ผ่านการกรองมัธยฐานด้วยขนาดหน้าต่าง 3x3 (ค)
ภาพ CMAN ที่ผ่านการกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ (ง)

ภาพที่ 6.1 (ต่อ)

(จ)

(ฉ)



(ช)

(ซ)

ภาพที่ 6.1 (ต่อ) ภาพ CMAN ที่ผ่านการกรองมัธยฐานมัลติเชลล์ (จ)

ภาพ CMAN ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ 1st algorithm (ฉ)

ภาพ CMAN ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ 2st algorithm Square shape (ช)

ภาพ CMAN ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลง 2st algorithm cross shape (ซ)

(ก)

(ข)

(ค)

(ง)

ภาพที่ 6.2 ภาพผลลัพธ์ของการกรอง ภาพ GIRL ต้นฉบับ (ก)
ภาพ GIRL ที่มีสัญญาณรบกวนอิมพัลส์ 5% (ข)
ภาพ GIRL ที่ผ่านการกรองมัธยฐานด้วยขนาดหน้าต่าง 3x3 (ค)
ภาพ GIRL ที่ผ่านการกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติ (ง)

ภาพที่ 6.2 (ต่อ)

(จ)

(ฉ)

(ช)

(ซ)

ภาพที่ 6.2 (ต่อ) ภาพ GIRL ที่ผ่านการกรองมัธยฐานมัลติเชลล์ (จ)
ภาพ GIRL ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ 1st algorithm (ฉ)
ภาพ GIRL ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ 2st algorithm Square shape (ช)
ภาพ GIRL ที่ผ่านการกรองมัธยฐานมัลติเชลล์แบบดัดแปลงที่ 2st algorithm cross shape (ซ)

ตารางที่ 6.1 เปรียบเทียบประสิทธิภาพการกรองชนิดต่าง ๆ ของภาพ CMAN กับค่า MSE และ MAE

IMAGE FILES	FILTER	MSE	MAE
CMAN	INPUT	1097.704	6.214
	MEDIAN 3x3	175.729	5.256
	ADAPTIVE WINDOWS SIZE	235.748	4.762
	MULTI-SHELL (2 SHELL)	101.189	1.508
	AMS 1st ALGORITHM	77.837	1.901
	AMS 2nd ALGORITHM SQUARE SHAPE	72.768	1.992
	AMS 2nd ALGORITHM CROSS SHAPE	72.147	1.913

ตารางที่ 6.2 เปรียบเทียบประสิทธิภาพการกรองชนิดต่าง ๆ ของภาพ GIRL กับค่า MSE และ MAE

IMAGE FILES	FILTER	MSE	MAE
GIRL	INPUT	987.518	6.16
	MEDIAN 3x3	31.029	2.346
	ADAPTIVE WINDOWS SIZE	34.923	1.652
	MULTI-SHELL (2 SHELL)	40.795	0.797
	AMS 1st ALGORITHM	29.966	0.892
	AMS 2nd ALGORITHM SQUARE SHAPE	11.22	0.751
	AMS 2nd ALGORITHM CROSS SHAPE	14.78	0.832

ตารางที่ 6.3 เปรียบเทียบประสิทธิภาพการกรองชนิดต่าง ๆ ของภาพ LENA กับค่า MSE และ MAE

IMAGE FILES	FILTER	MSE	MAE
LENA	INPUT	888.303	5.676
	MEDIAN 3x3	69.567	3.761
	ADAPTIVE WINDOWS SIZE	85.687	3.029
	MULTI-SHELL (2 SHELL)	56.898	1.118
	AMS 1st ALGORITHM	33.843	0.566
	AMS 2nd ALGORITHM SQUARE SHAPE	24.346	1.157
	AMS 2nd ALGORITHM CROSS SHAPE	40.568	1.436

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการวิจัย

การกรองมัธยฐานเป็นพื้นฐานอย่างหนึ่งของการประมวลผลสัญญาณภาพ ข้อดีคือสามารถกำจัดสัญญาณรบกวนแบบอิมพัลส์ได้ดี แต่การรักษาความคมชัดของโครงร่างภาพและขอบภาพอยู่ในระดับไม่ดีนัก พบว่ายิ่งถ้าใช้ขนาดหน้าต่างที่ใหญ่ขึ้นความคมชัดจะยิ่งสูญเสียไปมากและหน้าต่างการกรองขนาดใหญ่จะมีความสามารถในการกำจัดสัญญาณรบกวนได้มากขึ้น การกรองมัธยฐานจึงได้มีการประยุกต์และดัดแปลงเป็นเทคนิคการกรองวิธีต่าง ๆ มากมายจากนักวิจัยทั่วโลก การกรองมัธยฐานแบบปรับขนาดหน้าต่างอัตโนมัติได้ถูกนำเสนอขึ้นมาอีก 1 วิธี ที่ช่วยเพิ่มประสิทธิภาพการกรองมัธยฐาน ข้อดีของวิธีนี้ คือ ปรับขนาดหน้าต่างได้ตั้งแต่ 1 พิกเซล จนถึงใหญ่สุด 5x5 พิกเซล (Cross shape) ซึ่งเป็นการปรับขนาดตามปริมาณสัญญาณรบกวนที่เข้ามา การอินพุทภาพผลลัพธ์ที่ได้จะมีความสม่ำเสมอดีขึ้น (ค่า MAE ในตารางที่ 6.1-6.3) แต่วิธีการนี้จะมีอัลกอลิธึมที่ใช้ในการตรวจสอบสัญญาณรบกวนค่อนข้างมาก และต้องทำการกรองถึง 2 ครั้ง แบ่งเป็นการกรองอิมพัลส์แบบ salt noise การกรองอีกครั้งสำหรับอิมพัลส์แบบ paper noise วิธีการกรองมัธยฐานมัลติเชลล์ มีหลักการทำงานที่ช่วยให้ภาพผลลัพธ์มีคุณภาพดีขึ้น แต่ติดขัดตรงที่ไม่สามารถกำจัดสัญญาณรบกวนอิมพัลส์ได้หมด ถ้าพิกเซลกึ่งกลางเป็นอิมพัลส์และมีอิมพัลส์ปรากฏบริเวณเชลล์เพียง 1 พิกเซลต่อเชลล์

การกรองมัธยฐานมัลติเชลล์แบบดัดแปลงจะช่วยส่งเสริมให้การกรองมัธยฐานมัลติเชลล์มีประสิทธิภาพที่ดีขึ้นและแก้ไขข้อบกพร่องได้ คือ สามารถกำจัดสัญญาณรบกวนในหน้าต่างได้หมดหรือมากกว่าวิธีเดิม การกรองมัธยฐานมัลติเชลล์แบบดัดแปลงมี 2 แบบ โดยแบบที่ 1 เป็นการเลียนแบบการกรองมัธยฐานมัลติเชลล์ แต่เปลี่ยนการแทนค่าพิกเซลที่จุดกึ่งกลางด้วยค่าพิกเซล (มากที่สุดหรือน้อยที่สุด) ในแนวตั้งแทนค่า Max (s) หรือ Min (s) ส่วนแบบที่ 2 เป็นการดัดแปลงการกรองมัธยฐานมัลติเชลล์ร่วมกับการกรองมัธยฐานแบบหน้าต่างคงที่ธรรมดา ผลดีคือสามารถกำจัดสัญญาณรบกวนได้มาก ภาพผลลัพธ์มีความคมชัดสูง ในแง่ของการเปรียบเทียบดัชนีการวัดด้วยค่า MSE และ MAE พบว่ามีประสิทธิภาพที่สูงกว่าวิธีอื่น ๆ

7.2 ข้อแสนอแนะการวิจัย

แนวทางในการวิจัยและพัฒนาต่อไปจะต้องพิจารณาภาพอินพุทและออกแบบลักษณะหน้าต่างการกรองให้มีความเหมาะสมสอดคล้องกัน เพื่อที่จะให้ได้ผลลัพธ์ที่ดีและต้องกับความต้องการส่วนการพัฒนาไปเป็นฮาร์ดแวร์นั่นเราจะต้องใช้เทคนิคและวิธีการออกแบบวงจรร่วมขนาดใหญ่ (VLSI) ซึ่งจะสามารถประยุกต์ไปใช้อุปกรณ์หรือเครื่องใช้ไฟฟ้าที่มีการประมวลผลสัญญาณดิจิตอล เช่น โทรทัศน์ความละเอียดสูง (HDTV)

บรรณานุกรม

[1] J.W. Tukey, *Exploratory Data Analysis (preliminary ed.)*. Reading, MA : Addison-Wesley, 1971.

[2] B. Justusson, "Median filtering : *Statistical properties*," Roy.Inst.Technol., Stockholm, Sweden, Tech.Rep. TRITA-MAT-1979-11,Aug.1979.

[3] W.K. Pratt, *Digital image processing*, Wiley, 1978.

[4] B.I. Justusson, "Median filtering statistical properties" in *Two-dimensional digital signal processing II* , T.S. Huang editor, Springer Verlag, 1981.

[5] G.J. Yong, T.S. Huang, "The effect of median filtering in edge location estimation", *Computer Vision, Graphics and Image Proceeding*, Vol. 15, pp. 224-225, 1981.

[6] P. Zamperoni, "Feature extraction by rank-order filtering for image segmentation ", *International Journal of Pattern Analysis and Artificial Inteligence* , Vol.2, no.2, pp. 301-319, 1988.

[7] S.S. Perlman, S. Eisenhandler, P.W. Lyons, M.J. Shumila, "Adaptive median filtering for impulse noise elimination in real-time TV signals" *IEEE Transactions on Communications*, Vol. COM-35, no. 6, 646-652, June 1987.

[8] S.S. H. Naqvi, N.C. Gallagher, E.J. Coyle, "An application of median filter to digital TV", *Proc. 1986 IEEE Int. Conf. On Acoustics, Speech and Signal Proceeding*, Tokyo, Japan, 1986.

[9] L.R. Rabiner, M.R. Sambur, C.E. Schmidt, "Application of a nonlinear smoothing algorithm to speech proceeding", *IEEE Transactions on Acoustics, Speech and Signal Proceeding*, Vol. ASSP-23, pp. 552-557, Dec. 1975.

[10] G. Arce, N.C. Gallagher, " State description for the root-signal set of median filters", *IEEE Transactions on Acoustics, Speech and Signal Proceeding*, Vol. ASSP-30, no. 6, pp. 894-902, Dec. 1982.

[11] P. Chan, J.S. Lim, "One-dimensional proceeding for adaptive image restoration, "*IEEE Transactions on Acoustics, Speech and Signal Proceeding*, Vol. ASSP-33, no. 6, pp. 117-126, Feb. 1985.

[12] J.S.Jimmy Li, "A class of multi-shell min/max median filters," Proceedings of 1989 IEEE International Symposium on Circuits and Systems, Oregon, May, 1989.

[13] J.S.Jimmy Li and Stefan Luthi, "A real-time 2-D median based filter for video signals," *IEEE Transactions on Consumer Electronics, Vol.39, No.2, May, 1993*

[14] Ioanais Pitas, "Fast algorithms for running ordering and max/min calculation," *IEEE Transactions on* Circuits and Systems, Vol. CAS-36, June 1989, pp.795-804

ภาคผนวก

```
/*Program : MEDIAN.C
  Date    : 21 October 1999
  By      : Narupon Panakulchiwit
*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <alloc.h>
#include <ctype.h>

#define MAX 256
#define maxy 255
#define maxx 255
#define text "\nMalloc memory error\n"
#define uc unsigned char
#define ui unsigned int

void MemoryAllocate(void);
void OpenFile(void);
void InitialGraph(void);
void CrossMedianPlus(void);
void Median3x3(void);
void Average3x3(void);
void Average5x5(void);
void Median5x5(void);
void MSMedian(void);
void MSMedianforML(void);
void SISMedian(void);
void SIS_ML_Median(void);
void MMS2Median(void);
void AMS3Median(void);
void AMS4Median(void);
void ALMedian(void);
void ReleaseMemory(void);
void SaveImage(void);

uc far **pixel,**pixel2;
int i;

void MemoryAllocate(void)
{
 if((pixel = (uc **)farcalloc(sizeof(uc *),
(MAX+4)))==NULL) {
   printf(text);
   getch();
   exit(1);
  }
  for(i=0;i<MAX+4;i++)
   if((pixel[i] = (uc *)farcalloc(sizeof(uc),
(MAX+4)))==NULL)  {
     printf(text);
     getch();
     exit(1);
    }

  if((pixel2 = (uc **)farcalloc(sizeof(uc *),
(maxy+1)))==NULL) {
   printf(text);
   getch();
   exit(1);
  }
  for(i=0;i<maxx+1;i++)
   if((pixel2[i] = (uc *)farcalloc(sizeof(uc),
(maxx+1)))==NULL) {
     printf(text);
     getch();
     exit(1);
    }
}

void OpenFile(void) {
 int i,j;
 char filename[20];
 FILE *fin;

 clrscr();
 printf("PLEASE ENTER FILE NAME : ");
 gets(filename);
 if ((fin = fopen(filename,"rb"))==NULL) {
   printf("Error : Can't open image file!");
   for(i=0;i<MAX+4;i++)
    farfree((uc *)pixel[i]);
   farfree(pixel);
   exit(0);
 }

 /* read value image ;i=X axis, j=Y axis*/
 for(j=2;j<MAX+2;j++)
  for(i=2;i<MAX+2;i++)
   fread(&pixel[i][j],sizeof(uc),1U,fin);

 for(j=0;j<2;j++) //2 lines top
  for(i=2;i<MAX+2;i++)  //X axis
   pixel[i][j]=pixel[i][j+2];


 for(j=2+MAX;j<4+MAX;j++)     //2 lines
bottom
  for(i=2;i<MAX+2;i++)
   pixel[i][j]=pixel[i][j-2];

 for(j=2;j<MAX+2;j++)   //2 lines left
  for(i=0;i<2;i++)
   pixel[i][j]=pixel[i+2][j];

 for(j=2;j<MAX+2;j++)   //2 lines right
  for(i=2+MAX;i<4+MAX;i++)
```

```
    pixel[i][j]=pixel[i-2][j];
}
int huge DetectGraph256(void) {
 return 2;
}

void setrgb(uc i,uc r,uc g,uc b) {
 outp(0x3c8,i);
 outp(0x3c9,r);
 outp(0x3c9,g);
 outp(0x3c9,b);
}

void InitialGraph(void) {
 int gdriver, gmode, errorcode;
 gdriver =
installuserdriver("svga256",DetectGraph256);
 gdriver = DETECT;

 /* initialize graphics mode */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();

 if (errorcode != grOk)  /* an error occurred */ {
   printf("Graphics error: %s\n", grapherrormsg
(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);             /* return with error code */
 }
 /* create gray scale */
 for (i=0; i<256; i++)
   setrgb(i,i>>2,i>>2,i>>2);
}

void CrossMedianPlus() {
 int i,j,x,y;
 uc tmp,sort[5];

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y];
   sort[1] = pixel[x][y-1];
   sort[2] = pixel[x][y];
   sort[3] = pixel[x+1][y];
   sort[4] = pixel[x][y+1];
   for(j=0;j<4;j++)
        for(i=0;i<(4-j);i++)
         if(sort[i]>sort[i+1]) {
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   pixel2[x-2][y-2] = sort[2];       //Offset
  }

 for(j=0;j<MAX;j++)
  for(i=0;i<MAX;i++)
    putpixel(300+i,j,pixel2[i][j]);
}

void Median3x3(void) {
 int i,j,x,y;
 uc tmp,sort[9];

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y-1];
   sort[1] = pixel[x-1][y];
   sort[2] = pixel[x-1][y+1];
   sort[3] = pixel[x][y-1];
   sort[4] = pixel[x][y];
   sort[5] = pixel[x][y+1];
   sort[6] = pixel[x+1][y-1];
   sort[7] = pixel[x+1][y];
   sort[8] = pixel[x+1][y+1];
   for(j=0;j<8;j++)
        for(i=0;i<(8-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   pixel2[x-2][y-2] = sort[4];
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void Average3x3(void) {
 int i,j,x,y;
 uc tmp,sort[9];
 int Sum1;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y-1];
   sort[1] = pixel[x-1][y];
   sort[2] = pixel[x-1][y+1];
   sort[3] = pixel[x][y-1];
   sort[4] = pixel[x][y];
   sort[5] = pixel[x][y+1];
   sort[6] = pixel[x+1][y-1];
   sort[7] = pixel[x+1][y];
   sort[8] = pixel[x+1][y+1];
   Sum1=0;
   for(j=0;j<9;j++)
        Sum1= Sum1+sort[j];

   pixel2[x-2][y-2] = Sum1/9;
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void Average5x5(void) {
```

```
int i,j,x,y;
uc tmp,sort[25];
int Sum1;

for(y=2;y<MAX+2;y++)
 for(x=2;x<MAX+2;x++) {
  sort[0] =  pixel[x-2][y-2];
  sort[1] =  pixel[x-2][y-1];
  sort[2] =  pixel[x-2][y];
  sort[3] =  pixel[x-2][y+1];
  sort[4] =  pixel[x-2][y+2];
  sort[5] =  pixel[x-1][y-2];
  sort[6] =  pixel[x-1][y-1];
  sort[7] =  pixel[x-1][y];
  sort[8] =  pixel[x-1][y+1];
  sort[9] =  pixel[x-1][y+2];
  sort[10]=  pixel[x][y-2];
  sort[11]=  pixel[x][y-1];
  sort[12]=  pixel[x][y];
  sort[13]=  pixel[x][y+1];
  sort[14]=  pixel[x][y+2];
  sort[15] = pixel[x+1][y-2];
  sort[16] = pixel[x+1][y-1];
  sort[17] = pixel[x+1][y];
  sort[18] = pixel[x+1][y+1];
  sort[19] = pixel[x+1][y+2];
  sort[20] = pixel[x+2][y-2];
  sort[21] = pixel[x+2][y-1];
  sort[22] = pixel[x+2][y];
  sort[23] = pixel[x+2][y+1];
  sort[24] = pixel[x+2][y+2];

  Sum1=0;
  for(j=0;j<25;j++)
       Sum1= Sum1+sort[j];

  pixel2[x-2][y-2] = Sum1/25;
 }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void Median5x5() {
 int i,j,x,y;
 uc tmp,sort[25];

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] =  pixel[x-2][y-2];
   sort[1] =  pixel[x-2][y-1];
   sort[2] =  pixel[x-2][y];
   sort[3] =  pixel[x-2][y+1];
   sort[4] =  pixel[x-2][y+2];
   sort[5] =  pixel[x-1][y-2];
   sort[6] =  pixel[x-1][y-1];
   sort[7] =  pixel[x-1][y];
   sort[8] =  pixel[x-1][y+1];
   sort[9] =  pixel[x-1][y+2];
   sort[10]=  pixel[x][y-2];
   sort[11]=  pixel[x][y-1];
   sort[12]=  pixel[x][y];
   sort[13]=  pixel[x][y+1];
   sort[14]=  pixel[x][y+2];
   sort[15] = pixel[x+1][y-2];
   sort[16] = pixel[x+1][y-1];
   sort[17] = pixel[x+1][y];
   sort[18] = pixel[x+1][y+1];
   sort[19] = pixel[x+1][y+2];
   sort[20] = pixel[x+2][y-2];
   sort[21] = pixel[x+2][y-1];
   sort[22] = pixel[x+2][y];
   sort[23] = pixel[x+2][y+1];
   sort[24] = pixel[x+2][y+2];

       for(j=0;j<24;j++)
        for(i=0;i<(24-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   pixel2[x-2][y-2] = sort[12];
  }

 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void MSMedian() {
 int i,j,x,y;
 uc tmp,sort[8],mid;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] =  pixel[x-1][y-1];
   sort[1] =  pixel[x  ][y-1];
   sort[2] =  pixel[x+1][y-1];
   sort[3] =  pixel[x-1][y+1];
   sort[4] =  pixel[x  ][y+1];
   sort[5] =  pixel[x+1][y+1];
   sort[6] =  pixel[x-1][y  ];
   sort[7] =  pixel[x+1][y  ];
   mid     =  pixel[x][y];
   for(j=0;j<7;j++)
        for(i=0;i<(7-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   if    ( (sort[7]>mid)&&(mid>sort[0]) )
          pixel2[x-2][y-2] = mid;
   else if( (sort[7]>mid)&&(mid<sort[0]) )
          pixel2[x-2][y-2] = sort[0];
   else if( (sort[7]<mid)&&(mid>sort[0]) )
          pixel2[x-2][y-2] = sort[7];
   else pixel2[x-2][y-2] = mid;
  }
```

```
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void MSMedianforML() {
 int i,j,x,y;
 uc tmp,sort[6],mid;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y-1];
   sort[1] = pixel[x][y-1];
   sort[2] = pixel[x+1][y-1];
   sort[3] = pixel[x-1][y+1];
   sort[4] = pixel[x][y+1];
   sort[5] = pixel[x+1][y+1];
   mid   = pixel[x][y];
   for(j=0;j<5;j++)
        for(i=0;i<(5-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   if   ( (sort[5]>mid)&&(mid>sort[0]) )
        pixel2[x-2][y-2] = mid;
   else if( (sort[5]>mid)&&(mid<sort[0]) )
        pixel2[x-2][y-2] = sort[0];
   else if( (sort[5]<mid)&&(mid>sort[0]) )
        pixel2[x-2][y-2] = sort[5];
   else pixel2[x-2][y-2] = mid;
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void SISMedian() {              //Multi-shell
Shell In Shell
 int i,j,x,y;
 uc tmp,sort[8],mid,sorts2[16];

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y-1];
   sort[1] = pixel[x][y-1];
   sort[2] = pixel[x+1][y-1];
   sort[3] = pixel[x-1][y];
   sort[4] = pixel[x+1][y];
   sort[5] = pixel[x-1][y+1];
   sort[6] = pixel[x][y+1];
   sort[7] = pixel[x+1][y+1];
   mid   = pixel[x][y];
   for(j=0;j<7;j++)
        for(i=0;i<(7-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }

   sorts2[0] = pixel[x-2][y-2];
   sorts2[1] = pixel[x-1][y-2];
   sorts2[2] = pixel[x ][y-2];
   sorts2[3] = pixel[x+1][y-2];
   sorts2[4] = pixel[x+2][y-2];
   sorts2[5] = pixel[x-2][y-1];
   sorts2[6] = pixel[x+2][y-1];
   sorts2[7] = pixel[x-2][y ];
   sorts2[8] = pixel[x+2][y ];
   sorts2[9] = pixel[x-2][y+1];
   sorts2[10]= pixel[x+2][y+1];
   sorts2[11]= pixel[x-2][y+2];
   sorts2[12]= pixel[x-1][y+2];
   sorts2[13]= pixel[x ][y+2];
   sorts2[14]= pixel[x+1][y+2];
   sorts2[15]= pixel[x+2][y+2];
   for(j=0;j<15;j++)
        for(i=0;i<(15-j);i++)
         if(sorts2[i]>sorts2[i+1]){
          tmp = sorts2[i];
          sorts2[i] = sorts2[i+1];
          sorts2[i+1] = tmp;
         }

   //sort[0] = sort[0];
   sort[1] = sort[7];
   sort[2] = mid;
   sort[3] = sorts2[0];
   sort[4] = sorts2[15];
   for(j=0;j<4;j++)
        for(i=0;i<(4-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }

   pixel2[x-2][y-2] = sort[2];
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void SIS_ML_Median() {          //Multi-shell
Shell In Shell for Missing Line
 int i,j,x,y;
 uc tmp,sort[8],mid,sorts2[16];

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] = pixel[x-1][y-1];
   sort[1] = pixel[x ][y-1];
   sort[2] = pixel[x+1][y-1];
   sort[3] = pixel[x-1][y+1];
   sort[4] = pixel[x ][y+1];
   sort[5] = pixel[x+1][y+1];
   mid   = pixel[x][y];
```

```
    for(j=0;j<5;j++)
          for(i=0;i<(5-j);i++)
           if(sort[i]>sort[i+1]){
            tmp = sort[i];
            sort[i] = sort[i+1];
            sort[i+1] = tmp;
           }

    sorts2[0] =  pixel[x-2][y-2];
    sorts2[1] =  pixel[x-1][y-2];
    sorts2[2] =  pixel[x  ][y-2];
    sorts2[3] =  pixel[x+1][y-2];
    sorts2[4] =  pixel[x+2][y-2];
    sorts2[5] =  pixel[x-2][y-1];
    sorts2[6] =  pixel[x+2][y-1];
    sorts2[7] =  pixel[x-2][y+1];
    sorts2[8] =  pixel[x+2][y+1];
    sorts2[9] =  pixel[x-2][y+2];
    sorts2[10]=  pixel[x-1][y+2];
    sorts2[11]=  pixel[x  ][y+2];
    sorts2[12]=  pixel[x+1][y+2];
    sorts2[13]=  pixel[x+2][y+2];
    for(j=0;j<13;j++)
          for(i=0;i<(13-j);i++)
           if(sorts2[i]>sorts2[i+1]){
            tmp = sorts2[i];
            sorts2[i] = sorts2[i+1];
            sorts2[i+1] = tmp;
           }

    //sort[0] =  sort[0];
    sort[1] =  sort[5];
    sort[2] =  mid;
    sort[3] =  sorts2[0];
    sort[4] =  sorts2[13];
    for(j=0;j<4;j++)
          for(i=0;i<(4-j);i++)
           if(sort[i]>sort[i+1]){
            tmp = sort[i];
            sort[i] = sort[i+1];
            sort[i+1] = tmp;
           }

    pixel2[x-2][y-2] = sort[2];
   }
  for(j=0;j<256;j++)
   for(i=0;i<256;i++)
    putpixel(300+i,j,pixel2[i][j]);
}

void MMS2Median() {
 int i,j,x,y;
 uc tmp,sort[6],mid,re_max,re_min;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] =  pixel[x-1][y-1];
   sort[1] =  pixel[x][y-1];
   sort[2] =  pixel[x+1][y-1];
   sort[3] =  pixel[x-1][y+1];
   sort[4] =  pixel[x][y+1];
   sort[5] =  pixel[x+1][y+1];
   mid    =  pixel[x][y];
   if (sort[1]>sort[4]) {
        re_max=sort[1];
        re_min=sort[4];
   }
   else {
        re_max=sort[4];
        re_min=sort[1];
   }
   for(j=0;j<5;j++)
        for(i=0;i<(5-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   if(    (mid>=sort[5]) )          pixel2[x-2]
[y-2] = re_max;
   else if( (mid<=sort[0]) )
        pixel2[x-2][y-2] = re_min;
   else pixel2[x-2][y-2] = mid;
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void AMS3Median(){
 int i,j,x,y;
 uc tmp,sort[8],mid;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] =  pixel[x-1][y-1];
   sort[1] =  pixel[x  ][y-1];
   sort[2] =  pixel[x+1][y-1];
   sort[3] =  pixel[x-1][y];
   sort[4] =  pixel[x+1][y];
   sort[5] =  pixel[x-1][y+1];
   sort[6] =  pixel[x  ][y+1];
   sort[7] =  pixel[x+1][y+1];
   mid    =  pixel[x][y];
   for(j=0;j<7;j++)
        for(i=0;i<(7-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   if(    (sort[7]> mid)&&(mid<=sort[0]))
        pixel2[x-2][y-2] = sort[3];
   else if( (sort[7]<=mid)&&(mid> sort[0]))
        pixel2[x-2][y-2] = sort[4];
   else  //(sort[7]>=mid)&&(mid>=sort[0]))
        pixel2[x-2][y-2] = mid;
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
```

```
    putpixel(300+i,j,pixel2[i][j]);
}

void AMS4Median(){
 int i,j,x,y;
 uc tmp,sort[8],mid;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   sort[0] =  pixel[x-1][y];
   sort[1] =  pixel[x][y+1];
   sort[2] =  pixel[x+1][y];
   sort[3] =  pixel[x][y-1];
   sort[4] =  pixel[x-2][y];
   sort[5] =  pixel[x][y+2];
   sort[6] =  pixel[x+2][y];
   sort[7] =  pixel[x][y-2];

   mid    =  pixel[x][y];
   for(j=0;j<7;j++)
        for(i=0;i<(7-j);i++)
         if(sort[i]>sort[i+1]){
          tmp = sort[i];
          sort[i] = sort[i+1];
          sort[i+1] = tmp;
         }
   if(    (sort[7]> mid)&&(mid<=sort[0]))
        pixel2[x-2][y-2] = sort[3];
   else if( (sort[7]<=mid)&&(mid> sort[0]))
        pixel2[x-2][y-2] = sort[4];
   else  //(sort[7]>=mid)&&(mid>=sort[0]))
        pixel2[x-2][y-2] = mid;
  }
 for(j=0;j<256;j++)
  for(i=0;i<256;i++)
   putpixel(300+i,j,pixel2[i][j]);
}

void ALMedian(){
 int i,j,x,y,Lh=0,Lv=0,length;
 int P[4],Q[4];
 uc tmp,sort[9];
 int T1=15,T2=50;

 for(y=2;y<MAX+2;y++)
  for(x=2;x<MAX+2;x++) {
   for(j=1;j<3;j++)
        P[j+1]=(int)(pixel[x+j-1][y]-pixel[x+j]
[y]);    //P[2]=p1,P[3]=p2
   for(j=-2;j<0;j++)
        P[j+2]=(int)(pixel[x+j+1][y]-pixel[x+j]
[y]);  //P[1]=p-1,P[0]=p-2

   if( ((P[0]>T2)&&(P[3]>T2))||((P[0]>T2)&&
(P[2]>T2))||((P[1]>T2)&&(P[3]>T2)) ) {
        Lh=5;
        goto ALM1;
   }
   else if( (P[2]>T1)&&(P[1]>T1) ) {
        Lh=3;
        goto ALM1;
   }
   else if (P[1]!=0)  //Check low amplitude
impulse
   { if( ((float)(P[2]/P[1])>(0.5)) && ((float)(P
[2]/P[1])<(2.0)) )
        Lh=3;
        goto ALM1;
   }
   else if( ((P[1]>(-T1))&&(P[1]<T1)) && ((P
[2]>(-T1))&&(P[2]<T1)) &&
        ((P[0]>(-T2))&&(P[0]<T2)) && ((P
[3]>(-T2))&&(P[3]<T2)) ){
        Lh=1;
        Lv=1;
        goto ALM2;
   }
   else
        Lh=1;

ALM1:;
   for(j=1;j<3;j++)
        Q[j+1]=(int)(pixel[x][y+j-1]-pixel[x]
[y+j]);  //Q[2]=q1,Q[3]=q2
   for(j=-2;j<0;j++)
        Q[j+2]=(int)(pixel[x][y+j+1]-pixel[x]
[y+j]);  //Q[1]=q-1,Q[0]=q-2

   if( ((Q[0]>T2)&&(Q[3]>T2))||((Q[0]>T2)&&
(Q[2]>T2))||((Q[1]>T2)&&(Q[3]>T2)) ) {
        Lv=5;
        goto ALM2;
   }
   else if( (Q[2]>T1)&&(Q[1]>T1) ) {
        Lv=3;
        goto ALM2;
   }
   else if (Q[1]!=0)  //Check low amplitude
impulse
   { if( ((float)(Q[2]/ Q[1])>(0.5)) && ((float)(Q
[2]/ Q[1])<(2.0)) )
        Lh=3;
        goto ALM2;
   }
   else if( ((Q[1]>(-T1))&&(Q[1]<T1)) && ((Q
[2]>(-T1))&&(Q[2]<T1)) &&
        ((Q[0]>(-T2))&&(Q[0]<T2)) &&
((Q[3]>(-T2))&&(Q[3]<T2)) ){
        Lh=1;
        Lv=1;
        goto ALM2;
   }
   else
        Lv=1;
ALM2:;
 switch (Lh) {
        case 1: sort[0] = pixel[x][y];
        //1x(1-5)
         switch (Lv) {
```

```
        case 1: //sort[0] = pixel[x][y]; //1x1
                break;
        case 3: sort[1] = pixel[x][y-1]; //1x3
                sort[2] = pixel[x][y+1];
                break;
        case 5: sort[1] = pixel[x][y-1]; //1x5
                sort[2] = pixel[x][y+1];
                sort[3] = pixel[x][y-2];
                sort[4] = pixel[x][y+2];
                break;
               } break;

      case 3: sort[0] = pixel[x][y];  //3x(1-5)
                sort[1] = pixel[x-1][y];
                sort[2] = pixel[x+1][y];
       switch (Lv) {
        case 1: //sort[0] = pixel[x][y]; //3x1
                break;
        case 3: sort[3] = pixel[x][y-1]; //3x3
                sort[4] = pixel[x][y+1];
                break;
        case 5: sort[3] = pixel[x][y-1]; //3x5
                sort[4] = pixel[x][y+1];
                sort[5] = pixel[x][y-2];
                sort[6] = pixel[x][y+2];
                break;
               } break;

      case 5: sort[0] = pixel[x][y];  //5x(1-5)
                sort[1] = pixel[x-1][y];
                sort[2] = pixel[x+1][y];
                sort[3] = pixel[x-2][y];
                sort[4] = pixel[x+2][y];
       switch (Lv) {
        case 1: //sort[0] = pixel[x][y]; //5x1
                break;
        case 3: sort[5] = pixel[x][y-1]; //5x3
                sort[6] = pixel[x][y+1];
                break;
        case 5: sort[5] = pixel[x][y-1]; //5x5
                sort[6] = pixel[x][y+1];
                sort[7] = pixel[x][y-2];
                sort[8] = pixel[x][y+2];
                break;
               } break;
      default : sort[0] = pixel[x][y];   //1x1
}

if(Lh==1 && Lv==1)
      pixel2[x-2][y-2] =pixel[x][y];

else {
      length=(Lh+Lv-2)/2;
      for(j=0;j<(Lh+Lv-2);j++)
        for(i=0;i<(Lh+Lv-2-j);i++)
          if(sort[i]>sort[i+1]){
            tmp = sort[i];
            sort[i] = sort[i+1];
            sort[i+1] = tmp;
          }
            pixel2[x-2][y-2] = sort[length];
      }
    }

  for(j=0;j<256;j++)
    for(i=0;i<256;i++)
      putpixel(300+i,j,pixel2[i][j]);    //End salt
noise
//    getch();
//Start pepper noise
  for(j=0;j<maxy+1;j++)
          //Transfer first result data
    for(i=0;i<maxx+1;i++)
      pixel[i+2][j+2]=pixel2[i][j];

  for(y=2;y<MAX+2;y++)
    for(x=2;x<MAX+2;x++) {
      for(j=1;j<3;j++)
          P[j+1]=(int)(pixel[x+j-1][y]-pixel[x+j]
[y]);      //P[2]=p1,P[3]=p2
      for(j=-2;j<0;j++)
          P[j+2]=(int)(pixel[x+j+1][y]-pixel[x+j]
[y]);   //P[1]=p-1,P[0]=p-2

      if( ((-P[0]>T2)&&(-P[3]>T2))||((-P[0]>
T2)&&(-P[2]>T2))||((-P[1]>T2)&&(-P[3]>T2)) )
{
          Lh=5;
          goto ALM3;
      }
      else if( (-P[2]>T1)&&(-P[1]>T1) ) {
          Lh=3;
          goto ALM3;
      }
      else if (P[1]!=0)  //Check low amplitude
impulse
      { if( ((float)(P[2]/P[1])>(0.5)) && ((float)(P
[2]/P[1])<(2.0)) )
          Lh=3;
          goto ALM3;
      }
      else if( ((-P[1]>(-T1))&&(-P[1]<T1)) && ((-
P[2]>(-T1))&&(-P[2]<T1)) && ((-P[0]>(-T2))
&&(-P[0]<T2)) && ((-P[3]>(-T2))&&(-P[3]
<T2)) ){
          Lh=1;
          Lv=1;
          goto ALM4;
      }
      else
          Lh=1;

ALM3:;
      for(j=1;j<3;j++)
          Q[j+1]=(int)(pixel[x][y+j-1]-pixel[x]
[y+j]);   //Q[2]=q1,Q[3]=q2
      for(j=-2;j<0;j++)
          Q[j+2]=(int)(pixel[x][y+j+1]-pixel[x]
[y+j]);   //Q[1]=q-1,Q[0]=q-2
```

```
    if( ((-Q[0]>T2)&&(-Q[3]>T2))||((-Q[0]>
T2)&&(-Q[2]>T2))||((-Q[1]>T2)&&(-Q[3]>T2))
) {
        Lv=5;
        goto ALM4;
    }
    else if( (-Q[2]>T1)&&(-Q[1]>T1) ) {
        Lv=3;
        goto ALM4;
    }
    else if (Q[1]!=0)  //Check low amplitude
impulse
    { if( ((float)(Q[2]/Q[1])>(0.5)) && ((float)(Q
[2]/Q[1])<(2.0)) )
        Lh=3;
        goto ALM4;
    }
    else if( ((-Q[1]>(-T1))&&(-Q[1]<T1)) && ((-
Q[2]>(-T1))&&(-Q[2]<T1)) &&
            ((-Q[0]>(-T2))&&(-Q[0]<T2)) &&
((-Q[3]>(-T2))&&(-Q[3]<T2)) ){
        Lh=1;
        Lv=1;
        goto ALM4;
    }
    else
        Lv=1;

ALM4:;
    switch (Lh) {
        case 1: sort[0] = pixel[x][y];    //1x(1-5)
         switch (Lv) {
          case 1: //sort[0] = pixel[x][y]; //1x1
                  break;
          case 3: sort[1] = pixel[x][y-1]; //1x3
                  sort[2] = pixel[x][y+1];
                  break;
          case 5: sort[1] = pixel[x][y-1]; //1x5
                  sort[2] = pixel[x][y+1];
                  sort[3] = pixel[x][y-2];
                  sort[4] = pixel[x][y+2];
                  break;
        } break;

        case 3: sort[0] = pixel[x][y];    //3x(1-5)
                sort[1] = pixel[x-1][y];
                sort[2] = pixel[x+1][y];
         switch (Lv) {
          case 1: //sort[0] = pixel[x][y]; //3x1
                  break;
          case 3: sort[3] = pixel[x][y-1]; //3x3
                  sort[4] = pixel[x][y+1];
                  break;
          case 5: sort[3] = pixel[x][y-1]; //3x5
                  sort[4] = pixel[x][y+1];
                  sort[5] = pixel[x][y-2];
                  sort[6] = pixel[x][y+2];
                  break;
        } break;

        case 5: sort[0] = pixel[x][y];    //5x(1-5)
                sort[1] = pixel[x-1][y];
                sort[2] = pixel[x+1][y];
                sort[3] = pixel[x-2][y];
                sort[4] = pixel[x+2][y];
         switch (Lv) {
          case 1: //sort[0] = pixel[x][y]; //5x1
                  break;
          case 3: sort[5] = pixel[x][y-1]; //5x3
                  sort[6] = pixel[x][y+1];
                  break;
          case 5: sort[5] = pixel[x][y-1]; //5x5
                  sort[6] = pixel[x][y+1];
                  sort[7] = pixel[x][y-2];
                  sort[8] = pixel[x][y+2];
                  break;
         } break;
        default : sort[0] = pixel[x][y];    //1x1
    }

    if(Lh==1 && Lv==1)
        pixel2[x-2][y-2] =pixel[x][y];
    else {
        length=(Lh+Lv-2)/2;
        for(j=0;j<(Lh+Lv-2);j++)
         for(i=0;i<(Lh+Lv-2-j);i++)
          if(sort[i]>sort[i+1]){
           tmp = sort[i];
           sort[i] = sort[i+1];
           sort[i+1] = tmp;
          }
        pixel2[x-2][y-2] = sort[length];
       }
      }

    for(j=0;j<256;j++)
     for(i=0;i<256;i++)
      putpixel(300+i,j,pixel2[i][j]);
}

void ReleaseMemory(void) {
 closegraph();
 for(i=0;i<MAX+4;i++)  farfree((uc *)pixel[i]);
 farfree(pixel);
 for(i=0;i<maxx+1;i++)  farfree((uc *)pixel2[i]);
 farfree(pixel2);
}

void SaveImage() {
 unsigned char filename1[20];
 char ch1;
 int i,j;
 FILE *fout;

 ch1 = tolower(getch()); putch(ch1);
 if(ch1 == 's') {
  closegraph();
  printf("\nSave image");
  printf("\nPlease enter file name : ");
  gets(filename1);
```

```
  if ((fout = fopen(filename1,"wb"))==NULL) {
   printf("\nError : Can't save image file!");
  }
  else {
   for(j=0;j<maxy+1;j++)
        for(i=0;i<maxx+1;i++)
         fwrite(&pixel2[i][j],sizeof
(uc),1U,fout);
   printf("\nSave image file complete      ");
   exit(0);
  }
 }
 else if (ch1 == 'q') {
  closegraph();
  printf("\nQuit program");
  exit(0);
 }
}

void main(void)
{
 unsigned int i,j;
 clrscr();
 MemoryAllocate();
 OpenFile();
 InitialGraph();

 for(j=2;j<MAX+2;j++)
  for(i=2;i<MAX+2;i++)
   putpixel(i,j,pixel[i][j]);
 outtextxy(10,260,"ORIGINAL IMAGE");
 getch();

 CrossMedianPlus();
 setcolor(255);
 outtextxy(310,260,"Crossplus median filter");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"Crossplus median filter");

 Median3x3();
 setcolor(255);
 outtextxy(310,260,"3x3 MEDIAN FILTER");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"3x3 MEDIAN FILTER");

 Median5x5();
 setcolor(255);
 outtextxy(310,260,"5x5 MEDIAN FILTER");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"5x5 MEDIAN FILTER");

 Average3x3();
 setcolor(255);
 outtextxy(310,260,"3x3 Average FILTER");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"3x3 Average FILTER");

 Average5x5();
 setcolor(255);
 outtextxy(310,260,"5x5 Average FILTER");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"5x5 Average FILTER");

 ALMedian();
 setcolor(255);
 outtextxy(310,260,"Adaptive Windows Size
Median Filter");
 SaveImage();
 setcolor(0);
 outtextxy(310,260,"Adaptive Windows Size
Median Filter");

 MSMedian();
 setcolor(255);
 outtextxy(310,260,"MULTI SHELL MEDIAN
FILTER");
 SaveImage();
 setcolor(0);                    //Off label
name
 outtextxy(310,260,"MULTI SHELL MEDIAN
FILTER");

 MSMedianforML();
 setcolor(255);
 outtextxy(310,260,"Multi-shell median filter for
Missing line noise");
 SaveImage();
 setcolor(0);                    //Off label
name
 outtextxy(310,260,"Multi-shell median filter for
Missing line noise");

 SISMedian();
 setcolor(255);
 outtextxy(310,260,"MULTI SHELL (Shell in
shelll)");
 SaveImage();
 setcolor(0);                    //Off label
name
 outtextxy(310,260,"MULTI SHELL (Shell in
shelll)");

 SIS_ML_Median();
 setcolor(255);
 outtextxy(310,260,"MULTI SHELL (Shell in
shell for Missing Line Noise)");
 SaveImage();
 setcolor(0);                    //Off label
name
 outtextxy(310,260,"MULTI SHELL (Shell in
shell for Missing Line Noise)");
 MMS2Median();
       //Modify MSM 2
 setcolor(255);
```

```
  outtextxy(310,260,"Adaptive Multi-Shell 1st
algorithm");
  SaveImage();
  setcolor(0);                          //Off label
name
  outtextxy(310,260,"Adaptive Multi-Shell 1st
algorithm");

  AMS3Median();
  setcolor(255);
  outtextxy(310,260,"AMS 2nd algorithm square
shape");
  SaveImage();
  setcolor(0);                          //Off label
name
  outtextxy(310,260,"AMS 2nd algorithm square
shape");

  AMS4Median();
  setcolor(255);
  outtextxy(310,260,"AMS 2nd algorithm cross
shape");
  SaveImage();
  setcolor(0);                          //Off label
name
  outtextxy(310,260,"AMS 2nd algorithm cross
shape");

  ReleaseMemory();
}
```

ประวัติผู้เขียน

นายนฤพนธ์ พนากุลชัยวิทย์ เกิดเมื่อวันที่ 5 ตุลาคม 2513 ที่จังหวัดชลบุรี สำเร็จการศึกษาปริญญาตรีอุตสาหกรรมศาสตรบัณฑิต (สาขาเทคโนโลยีการวัดคุมทางอุตสาหกรรม) คณะวิศวกรรมศาสตร สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2535 และประกาศนียบัตรวิชาชีพชั้นสูง (อิเล็กส์ทรอนิกส์) จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าพระนครเหนือ ปีการศึกษา 2533

ในระหว่างการศึกษาระดับปริญญาโท ได้รับทุนการศึกษาคอมพิวเตอร์และการสื่อสารประจำปีการศึกษา 2536 จากบริษัท NEC (ประเทศไทย)

ปี พ.ศ. 2536 เข้าทำงานในตำแหน่งวิศวกรฝ่ายวิจัยและพัฒนาผลิตภัณฑ์ Remote control for air conditioner ที่บริษัทเมเจอร์อินดัสตรี จำกัด ปัจจุบันดำรงตำแหน่งหัวหน้าแผนกวิศวกรรมของบริษัทเมเจอร์อินดัสตรี จำกัด

ผลงานวิจัย

[1] นฤพนธ์ พนากุลชัยวิทย์ กอบชัย เดชหาญ สมยศ จุณณะปิยะ บัณฑรู พิทักษ์วงศ์ สมนึก ญาติปราโมทย์ วิษณุ กอพยัคฆินทร์ "การกรองสัญญาณมัธยฐานแบบมัลติเชลล์และแบบปรับขนาดหน้าต่างได้อย่างอัตโนมัติ," การประชุมวิชาการวิศวกรรมไฟฟ้า ครั้งที่ 18 จัดโดยมหาวิทยาลัยเทคโนโลยีมหานคร ณ.แอมบาสเดอร์ซิตี้ จอมเทียน พัทยา หน้า 921-925 2538

[2] **K. Dejhan**, N. Panakulchaiwit, D. Lisawatdiratanakul, F. Cheevasuvit, C. Soonyeekan and E. Prommas, "Adaptive multi-shell median filter for improving image quality," Proc. of the 18th Asian Conference on Remote Sensing (ACRS'97), pp.I-4-1-I-4-6, Kuala Lumpur, Malaysia, October 20-25, 1997.